“DYNAMIC LINKED LIBRARIES”:

Paradigms of the GPL license in contemporary software

LUIS ENRIQUEZ A.

LICENSE

ACKNOWLEDGMENTS

I would like to thank Dr. Till Jaeger for his important remarks, Dr. Axel Metzger and the IRI institute at the University of Hannover for all the support.

DEDICATION

Dedicated to Séverine and Tabata.

PREFACE

Free Software has transformed life on the planet. Since the appearance of the General Public License in 1989, traditional copyright theories have been challenged by the famous *copyleft.* These changes have become very important in the last years, because of the exponential increase of *free software* use in all developmental areas. However, there are still some gray areas of the GPL license and one of them is *dynamic linked libraries*.

License compatibility has become a real challenge because of the GPL license copyleft restrictions. The *Free Software Foundation* provides an interpretation of the GPL license in the field of linking program libraries applying different criteria such as modification, dependency, interaction, distribution medium, and location. These criteria goes beyond the traditional *derivative works* framework established in national copyright laws, and international copyright conventions. However, the GPL license as a copyright license has to be interpreted by Judges of different jurisdictions and different legal traditions.

The GPL FAQ interpretation might not be clear enough, and has created a lot uncertainty between developer communities and users. The practical relevance of this topic is huge, considering that the GPL license is the most popular FOSS license, and in contemporary software, computer programs use many dynamic libraries with the purpose of obtain or extend their functionality. Thus, license compatibility has become a real challenge because of the GPL license copyleft restrictions.

The answer is not black or white. The key should be to understand the GPL provisions in the light of a technical and legal analysis, and confront them to legal precedents and developer community disputes. Some of those controversies have been included in this work, with the purpose of applying the GPL interpretations over real cases.

What is the relation between copyleft and derivative works? How does the GPL FAQ

interpret them? Are these prescriptions regulated by applicable copyright law? Should copyright law be updated? Is copyright law prepared to regulate generic purpose licenses? How do we apply linking exceptions? Are the GPL prescriptions proportional? How would they fall into fair use exceptions? Should the Free Software Foundation be more flexible about their dynamic libraries linking interpretations? These and other questions will be covered in this work.

CONTENTS

CHAPTER ONE: DERIVATIVE WORKS LEGAL FRAMEWORK

This is an introductory chapter about the legal framework of derivative works. From a general perspective we must understand the dynamic linked libraries controversy as a derivative works controversy. The scope of derivative works is defined in different legal sources such as International Conventions, European Directives, and National Copyright Laws. Other relevant legal features such as computer programs interaction, and fair use exceptions, will also be covered here. Other GPL legal controversies outside the scope of derivative works, such as *license distribution*[1] or *license enforcement*[2], will not be covered here.

This chapter should provide a good legal background knowledge in the field of the *derivative works*, which is fundamental in order to understand the next chapters.

1.1. INTERNATIONAL CONVENTIONS AND DERIVATIVE WORKS

What is a derivative work? In order to answer this question we must refer to legal definitions under relevant Copyright International Conventions and Treaties[3]. The most relevant are The Revised Berne Convention[4], and the WIPO Copyright Treaty[5].

(1) The Revised Berne Convention. The most relevant international copyright convention is still the *Berne Convention for literary and artistic works of 1886*. This

1 Restrictions such as source code distribution and distribution of the license within the software are big legal issues by themselves, and that is why they are out of the scope of this work. This work is focused on derivative works and dynamic linked libraries paradigms.

2 License enforcement is another huge legal issue that won't be covered in this work. Nevertheless, sometimes it will be necessary to superficially refer to license enforcement when those are connected to the dynamic linking libraries paradigms.

3 International Treaties and Conventions have to be ratified by national parliaments in order to become a primary source of law. Treaties and Conventions have the same legal effect under International Law. The United Nations *Viena Convention on the Law of Treaties* of 1969 constitutes an important source of Treaty Law in our days. For more information about Treaty Law, see: http://www.treatylaw.org/whatisatreaty.asp.

4 See, http://www.wipo.int/treaties/en/ip/berne/trtdocs_wo001.html.

5 See, http://www.wipo.int/treaties/en/ip/wct/.

Convention has been signed and ratified by 166 countries[6], and has influenced most national copyright laws around the planet. Certainly there were not computer programs in 1886, and the original version of 1886 did not include scientific works[7]. The notion of digital copying and publishing was certainly not yet understood. But the Convention was revised a few times, and finally amended in 1979[8]. The last amended version of the Berne Convention included scientific works as subject of copyright protection:

> *"The expression "literary and artistic works" shall include every production in the literary, **scientific** and artistic domain, whatever may be the mode or form of its expression, such as books, pamphlets and other writings; lectures, addresses, sermons and other works of the same nature; dramatic or dramatico-musical works; choreographic works and entertainments in dumb show; musical compositions with or without words; cinematographic works to which are assimilated works expressed by a process analogous to cinematography; works of drawing, painting, architecture, sculpture, engraving and lithography; photographic works to which are assimilated works expressed by a process analogous to photography; works of applied art; illustrations, maps, plans, sketches and three-dimensional works relative to geography, topography, architecture or science"*[9].

As we can see, computer programs are not directly described, but they are included, because computer programs are *scientific productions.* The first computer appeared in 1946[10], and the term software appeared in the late 1950s[11], but the notion of software as a copyright subject came later. In the first computer years, a computer program was considered part of a hardware, and they were very few software producers. Between the late 1970s and the early 1980s, due to the spread of software production as an independent component of hardware, software was included by most legislations as subject of copyright law[12].

---

6 See, http://www.wipo.int/treaties/en/ShowResults.jsp?treaty_id=15.

7 Article 4 of the *Berne Convention of 1886* about scientific works as subject of copyright protection was more focused in scientific books, rather than software: "*...in fact, every production whatsoever in the literary, scientific, or artistic domain which can be published by any mode of **impression or reproduction***".

8 The original Berne Convention, the later revisions, and the amended version of 1979 are available at: http://keionline.org/copyright/berne.

9 See, Berne Convention art 2.1. Available at http://www.wipo.int/treaties/en/ip/berne/trtdocs_wo001.html.

10 Most believe the ENIAC was the first prototype of computer. See, http://ftp.arl.mil/mike/comphist/eniac-story.html.

11 For Software history, see: http://www.randomhistory.com/2008/06/26_software.html.

12 A recommended lecture about the history of copyright in Computer Software in the United States of America is: Garren Scott, *Copyright Protection of Computer Software: History, Politics, and Technology*, Massachusetts Institute of Technology, United States, 1991.

Software and computer programs are often referred as the same, but there is a little difference between them. A *computer program* is always *software*, but *software* is rather considered as a collection of programs, procedures, algorithms, concerning with the operation of a data processing system. Thus, the scope of software is wider than the computer program scope[13].

The Revised Berne Convention defines derivative works as: *"Translations, adaptations, arrangements of music and other alterations of a literary or artistic work shall be protected as original works without prejudice to the copyright in the original work"*[14].

This definition provides some criteria about what is a derivative work. Derivative works are allowed if they have their own expression of creativity. But when we adopt this definition to computer programs, some environmental conditions that artistic works don't have, might appear. For example, a restriction on the *right of use*: it is not forbidden to listen a song[15], but it might be forbidden to use software[16]. It is also possible to create a derivative work of a song based on listening to it, but it is not possible to create a derivative work of a computer program just by using it, without the source code.

(2) WIPO Copyright Treaty of Geneva 1996. The World Intellectual Property Organization copyright treaty provides more protection in the field of Copyright. It was signed in 1996 and has been contracted by 90 parties[17]. The Treaty does not provide a new definition for derivative works, but it adds the category of *computer programs* as subject of copyright protection.

The Treaty determines that the *expression* is the main criteria for copyright protection: "*Copyright protection extends to expressions and not to ideas, procedures, methods of operation or mathematical concepts as such*"[18]. Expression is the criteria for

13 For the purposes of this work, the terms *Software* and *Computer Programs* are used as synonyms.
14 See, Berne Convention art 2.3. Available at http://www.wipo.int/treaties/en/ip/berne/trtdocs_wo001.html#P85_10661.
15 No one can prohibit to hear or to watch. The legal issues come with distribution.
16 Such restriction is very common in proprietary software because a copy is needed for installation.
17 For more detail about contracting parties, see: http://www.wipo.int/treaties/en/ShowResults.jsp?lang=en&treaty_id=16.
18 See, WIPO art 2. Available at http://www.wipo.int/treaties/en/ip/wct/trtdocs_wo033.html#P136_19843.

establishing what is a derivative work. For instance, computer programs can be considered as an expression of the programmer. Programming requires creativity, and that creativity is expressed within the program. This could be considered the limit between computer programs and programming languages. A programming language[19] is a translator between the human programmer and the machine, so it doesn't have expression by itself, unless the human *programmer* creates something with it[20].

The protection for computer programs is the same as for literary works: *"Computer programs are protected as literary works within the meaning of Article 2 of the Berne Convention. Such protection applies to computer programs, whatever may be the mode or form of their expression"*[21].

As it was already mentioned, in 1886 they were not computer programs, and because of that, the WIPO Copyright Treaty clarifies that computer programs have the same level of protection in accordance to the ones described in article 2.1 of the Berne Convention.

1.2. EUROPEAN UNION LAW

In European secondary law[22], two European Directives have been created with the purpose of regulating the copyrights on Software, the Directive 91/250/EEC[23] and the Directive 2009/24/EC[24]. The latest one is a replacement of the former, which provides important improvements in the subject.

19 *"A programming language is a set of commands, instructions, and other syntax use to create a software program. Languages that programmers use to write code are called "high-level languages." This code can be compiled into a "low-level language," which is recognized directly by the computer hardware"*. See, http://www.techterms.com/definition/programming_language.

20 This affirmation is not absolute. Creating the basic libraries of programming languages also requires creativity.

21 See, WIPO art 4. Available at http://www.wipo.int/treaties/en/ip/wct/trtdocs_wo033.html#P136_19843.

22 *"The directive forms part of the secondary law of the European Union (EU). It is therefore adopted by the European institutions in accordance with the founding Treaties. Once adopted at European level, the directive is then transposed by Member States into their internal law"*. See, http://europa.eu/legislation_summaries/institutional_affairs/decisionmaking_process/l14527_en.htm.

23 See, http://www.wipo.int/wipolex/en/details.jsp?id=1424.

24 For more information on the creating process, see: http://europa.eu/legislation_summaries/internal_market/businesses/intellectual_property/mi0016_en.htm.

Directive 2009/24/EC on the legal protection of computer programs. Also known as the *Computer Program Directive*, it has the purpose of regulating computer programs in the European Union. It was written in accordance to the Berne Convention, and inspired by the Directive 91/250/EEC[25]. It goes further than the Berne Convention and the WIPO Copyright Treaty, because it regulates some particular aspects of software.

The directive does not provide a new derivative works definition, it rather follows the Berne Convention prescriptions. However, The criterion for copyright protection is wider than the international copyright conventions: *"A computer program shall be protected if it is original in the sense that it is the author's own intellectual creation. No other criteria shall be applied to determine its eligibility for protection"*[26].

Intellectual creation is established as the only criterion for copyright protection. This new perspective is certainly an improvement in order to avoid dealing with different interpretations of *expression.* Nevertheless, there is a restriction: *"...Ideas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright under this Directive"*[27].

The copyright holder's permission is disposed as the only legal way to authorize the creation of a derivative work: *"Article 2 shall include the right to do or to* ***authorise****: the translation, adaptation, arrangement and any other alteration of a computer program and the reproduction of the results thereof, without prejudice to the rights of the person who alters the program"*[28].

If the right holder doesn't provide the right of use and the right of modification, a derivative work could be considered copyright infringement. This provision certainly differs from the derivative works provision established in article 2.3 of the Berne Convention in relation to adaptations, transformations and musical arrangements[29].

25 Available at: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31991L0250:FR:HTML.
26 Art 1.3 Directive 2009/24/EC on the legal protection of computer programs.
27 Art 1.2 Directive 2009/24/EC on the legal protection of computer programs.
28 See, art 4.1(b) directive 2009/24/EC.
29 Once again, Software is different than artistic works. The article 2.3 of the Berne Convention is clearly focused on

The Directive also establishes the possibility of exceptions: *"In the absence of specific contractual provisions, the acts referred... shall not require authorization by the rightholder where they are necessary for the use of the computer program by the lawful acquirer..."*[30]. A *lawful acquirer* is someone who acquires the software with good faith and fair purposes[31].

Interaction is other relevant issue that the Directive regulates:

> *"The function of a computer program is to communicate and work together with other components of a computer system.... The parts of the program which provide for such interconnection and interaction between elements of software and hardware are generally known as interfaces. This functional interconnection and interaction is generally known as interoperability"*[32].

Interoperability is granted by the Directive. Nevertheless, the scope of interfaces is more complicated, and it will be tackled later on, when discussing the dynamic linked libraries paradigms.

Finally, the Directive provides an unclear exception:

> *"the person having a right to use a copy of a computer program shall be entitled, without the authorization of the right holder, to observe, study or test the functioning of the program in order to determine the ideas and principles which underlie any element of the program if he does so while performing any of the acts of loading, displaying, running, transmitting or storing the program which he is entitled to do"*[33].

In order to study the ideas and principles of the program, source code must be provided. Software developers know that loading, displaying and running, is not always enough, in order to study a program.

1.3. NATIONAL COPYRIGHT LAW

National Copyright Law have been widely influenced by International Conventions such

artistic works such as music.

30 See, art 5.1 directive 2009/24/EC.

31 A legal precedent in this field is *UsedSoft GmbH v. Oracle Intl. Corp* . Available at: http://curia.europa.eu/juris/document/document.jsf?docid=124564&doclang=en.

32 See, Recital 10 of the directive 2009/24/EC.

33 See, art 5.3 of directive 2009/24/EC.

as the Berne Convention and the WIPO Copyright Treaty. That is the reason why countries have adopted homogeneous legislations in many aspects such as copyright protection and derivative works. But they are other legal issues that would change radically the way copyright law is interpreted by different courts in different jurisdictions such as *fair use* exceptions.

Making an extended comparison between different national copyright laws is not the purpose of this work[34], so just few derivative work definitions will be taken.

(1) United States of America. The Copyright Act of 1976 was widely influenced by the Berne Convention. Derivative works are defined as:

> *"A "derivative work" is a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, **as a whole**, represent an original work of authorship, is a derivative work"*[35].

The notion of a *whole* is provided here. In the field of computer programs, the notion of what is a *whole* acquires a meaningful dimension, considering other components that might integrate a *whole*, such as the program libraries.

In US law, a relevant legal doctrine applied for copyright exceptions is the *fair use* doctrine: "*a fair use is any copying of copyrighted material done for a limited and transformative purpose, such as to comment upon, criticize, or parody a copyrighted work. Such uses can be done without permission from the copyright owner..."*[36]. The scope of *fair use* is not very clear, and must be interpreted in a case by case basis.

The Copyright Act of 1976[37] establishes four criteria to be considered for interpretation:

> *"1. the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;*
> *2. the nature of the copyrighted work;*
> *3. the amount and substantiality of the portion used in relation to the copyrighted work*

34 A recommended book which compiles different aspects of national copyright laws is: Van den Branden, Coughlan, Jaeger, *The International Free and Open source Book*, Open source Press, Germany, 2011.

35 See, 17 U.S.C. & 101 – Definitions. Available at http://www.law.cornell.edu/uscode/text/17/101

36 Definition transcribed from: http://fairuse.stanford.edu/Copyright_and_Fair_Use_Overview/chapter9/9-a.html

37 See, Copyright Act of 1976, 17 U.S.C., § 107.

as a whole;
4. the effect of the use upon the potential market for or value of the copyrighted work"[38].

(2) Germany. The German Copyright Law was published in 1965. It contains a definition of derivative works which also follows the Berne Convention:

> "*Translations and other adaptations of a work which constitute personal intellectual creations of the adapter shall enjoy protection as independent works without prejudice to copyright in the work that has been adapted.* ***Insignificant adaptations*** *of a non-protected musical work shall not enjoy protection as independent works"*[39].

Insignificant adaptations of musical works are not considered independent works.

(3) France. Copyright law is regulated in France by the *code de la propiété intellectuelle.* Derivative works are defined as:

> *"The authors of translations, adaptations, transformations or arrangements of works of the mind shall enjoy the protection afforded by this Code, without prejudice to the rights of the author of the original work. The same shall apply to the authors of anthologies or collections of miscellaneous works or data, such as databases, which, by reason of the selection or the arrangement of their contents, constitute intellectual creations..."*[40].

European countries also have their own *fair use* exceptions, but they are not fully harmonized. Harmonizing fair use exceptions were one of the purposes of the *Directive 2001/29/EC on the harmonization of certain aspects of copyright and related rights*[41]. Most common fair use exceptions between European countries are: "*private copy or other private use, parody, quotation, use of a work for scientific or teaching purposes, news reporting, library privileges, needs of the administration of justice and public policy"*[42]. But not all European countries recognize all of them, or some European countries have their own *fair use* exceptions[43].

38 See, Copyright Act of 1976, 17 U.S.C., § 107.
39 See, art 3 § 69c UrhG. Translation available at: http://www.iuscomp.org/gla/statutes/UrhG.htm#3.
40 See, Art L11203 Code de la propiété intellectuelle. Translation available at: http://www.wipo.int/wipolex/en/text.jsp?file_id=180336.
41 Exceptions and limitations are established in Article 5 of the Directive 2001/29/EC.
42 Dusollier Severine, *Fair Use by design in the European Copyright Directive of 2001: An Empty Promise*, University of Namur, Belgium. 2003. Page 3.
43 This *fair use* scenario gets much worst in an international context. An important copyright case in this field is the *Asphalt Jungle case*. The *Cour de cassation of France* applied a french public policy exception. See,

1.4. BRIEF GNU AND GPL HISTORY

The GNU[44] project appeared in 1984 with the purpose to create a free operating system which provides developers and users the freedom of using and modifying. The origin of free software seems to be a well know tale between Richard Stallman[45] and a printer at the MIT[46]. The MIT was the center of attention of hacker communities at the time, and Stallman was one of the most prominent. Stallman sent a 40 pages work to the printer, but the printer did not work. Stallman was used to repair such kind of miss configurations of old printers, so once again he tried to port his code in order to solve the problem of the new Xerox laser printer machine at the MIT[47].

But the operational software was distributed as binary code, and no help files were provided with the hardware. After a long sequence of intents, Stallman found an engineer of the Xerox Laser project at Carnegie Mellow Campus[48] in Palo-Alto California. After discussing with him, Stallman mentioned: "*He told me that he had promised not to give me a copy*"[49].

The GNU project was founded in 1984. GNU means *GNU's not Unix*, and it has developed the GNU Operating System since then. An operating system is: *"software that communicates with the hardware and allows other programs to run"*[50]. Well known tools and libraries developed by GNU were the GNU Emacs editor[51], the GNU C

http://www.juricaf.org/arret/FRANCE-COURDECASSATION-19910528-8919522.

44 *"GNU's not Unix"*. See, http://www.gnu.org/gnu/manifesto.html.

45 See, http://en.wikipedia.org/wiki/Richard_Stallman.

46 Massachusetts Institute of technology. See, http://www.mit.edu/.

47 See, Williams Sam, *Free as Freedom: Richard Stallman's crussade for Free Software*, Project Gutemberg eBook, 2002. Page 6.

48 See, http://www.cmu.edu/about/visit/campus-map.shtml.

49 See, Williams Sam, *Free as in Freedom: Richard Stallman's crussade for Free Software*, Project Gutemberg eBook, 2002.

50 Relevant components of an operating system are: System libraries, Programming language editors, interfaces, system utilities, device drivers, and even kernels. See, http://www.techterms.com/definition/operating_system.

51 *"GNU Emacs is an extensible, customizable text editor"*. See, http://www.gnu.org/software/emacs/.

library[52], the GNU make[53], the GNU Compiler collection GCC[54], the GNU Debugger GDB[55], the GNU Binutils[56], amongst many others.

The *Free Software Foundation* was created in 1985 as a nonprofit organization which institutionalized the Free Software philosophy. The FSF considers that *"Free software is a matter of liberty, not price"*[57]. Four freedoms are essential for free software[58]:

(0) Freedom to run the program for any purpose.

(1) Freedom to study how the program works and change it. Access to the source code is needed.

(2) Freedom to redistribute copies.

(3) Freedom to modify the software and distribute modified versions with the source code.

The right to produce derivative works is included in the third freedom.

The *GNU General Public License* was created by Richard Stallman in 1989 and it emerges as a result of all developments of the GNU project in the eighties. An important event that led Stallman to create the license was a conflict with *Unipress*[59], an enterprise which bought James Gosling's[60] rights of some C libraries for Emacs[61]. Stallman had to replaced them, creating the GNU Emacs. He decided to create a legal document to

52 *"Any Unix-like operating system needs a C library: the library which defines the ``system calls'' and other basic facilities such as open, malloc, printf, exit..."*. See, http://www.gnu.org/software/libc/.

53 *"Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files"*. See, http://www.gnu.org/software/make/.

54 *"The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstd++, libgcj,...). GCC was originally written as the compiler for the GNU operating system"*. See, http://gcc.gnu.org/.

55 *"GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed"*. See, http://www.gnu.org/software/gdb/.

56 *"The GNU Binutils are a collection of binary tools. The main ones are:* ***ld*** *- the GNU linker,* ***as*** *- the GNU assembler"*. See, http://www.gnu.org/software/binutils/.

57 See, http://www.fsf.org/about/.

58 See, http://www.gnu.org/philosophy/free-sw.html.

59 Unipress emacs no longer exists. It was replaced by GNU Emacs. See, http://emacswiki.org/emacs/jfm3.

60 See, http://en.wikipedia.org/wiki/James_Gosling.

61 Powerful text editor developed by Richard Stallman. See, http://gnu.org/software/emacs.

prevent free code from being proprietary[62]. The *Emacs General Public License*[63] establishes the idea of *copyleft*, and is the predecessor of the GPL license.

The GPL license came into existence as it is today in January 1989 as a revolutionary copyright license. It was described in the the GNU bulletin, the *January 1989 issue* as a new way of licensing systems. Stallman wrote:

> *"In the past, each copylefted program had to have its own copy of the General Public License contained in it... To make it easier to copyleft programs, we have been improving on the legalbol architecture of the General Public License to produce a new version that serves as a general-purpose subroutine: it can apply to any program without modification, no matter who is publishing it. All that's needed is a brief notice in the program itself, to say that the General Public License applies. Directions on doing this accompany the General Public License, so you can easily copyleft your programs"*[64].

The GPL v1 license established an innovative general purpose way of controlling derivative works, for copylefted software. In simple words, c*opyleft* means that a derivative work has to be licensed under the same license of its former work. This means that any software which has been programmed using GPL licensed software, would have to be licensed under the GPL license, as a viral effect that avoids the possibility of licensing.

The GNU General Public License v2 was released in 1991. This version clarified some GPL v1 issues, and has been by far the most popular of FOSS[65] licenses. The Linux kernel was released under this license in 1992 by *Linus Torvalds*[66]. This fact certainly increased the popularity of the license, and from that point, most free software has been released under the GPL v2.

Finally, the GNU General Public License v3 was released the 29 of June of 2007. It

62 See, http://free-soft.org/gpl_history.
63 See; http://www.free-soft.org/gpl_history/emacs_gpl.html.
64 See, http://free-soft.org/gpl_history.
65 *"Free and Open Source Software"*. See, http://en.wikipedia.org/wiki/Free_and_open_source_software.
66 See, http://fr.wikipedia.org/wiki/Linus_Torvalds.

added the *propagate* and *convey* definitions as replacement of the word *distribute*[67]. It also adds some restrictions in the use of DRMs[68], patents, and new definitions for *modifications* and *source code*[69].

1.5. GPL v2 AND DERIVATIVE WORKS

The GPL is a copyright license, and because of that, it has to be understood in relation of relevant legal issues such as jurisdiction, applicable law, and enforcement. These legal issues are normally established in proprietary licenses, but in a generic-purpose license such as the GPL, the establishment of these issues gets more complicated. A generic-purpose public license need to be adapted and interpreted by different *applicable laws*, *jurisdictions* and *legal systems*. If you ask a lawyer: *I want to claim copyright infringement?* He will reply: *Copyright under which applicable law?*

The GPL v2 license does not conceive *derivative works* in the same manner as national copyright laws and international copyright conventions. It rather has an independent and particular understanding of what derivative works are, in relation to an extended criteria such as modification, dependency, interaction, distribution medium, and location[70]. The GPL v2 establishes:

> *"You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions... These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works..."*[71].

The GPL license establishes a new an independent derivative works framework, which

67 See, http://www.gnu.org/licenses/gpl-faq.html#WhyPropagateAndConvey.
68 Digital Rights Management. See, http://en.wikipedia.org/wiki/Digital_rights_management.
69 A recommended review on GPL v3 improvements is available at: http://www.ifross.org/en/what-difference-between-gplv2-and-gplv3.
70 These criteria will be deeply analyzed in chapter 3 of this work.
71 See, GPL v2 section 2.0, paragraph 1, and 2.

would have to be applied for distribution of GPL works: "*Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program*"[72].

The *copyleft* is established: "*You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License*"[73].

This statement makes very clear that a derivative work based in a former work licensed by GPL v2 license, must also be licensed under the GPL v2 license. If we consider that in software, everything is usually built on top of other software, the *copyleft* requirement gets somehow difficult to determine. Therefore, it is precise to determine from a general perspective the kind of derivative works that are or not affected by the *copyleft*:

(1) Artistic Works v Computer programs. If you use GPL licensed software to create an artistic work such as music, video, or graphics, the artistic product is not considered a derivative work of that software. The copyleft only affects software. The FSF establishes: "*You can apply the GPL to any kind of work, as long as it is clear what constitutes the "source code" for the work*"[74]. This interpretation also applies to the GPL v3 license[75].

(2) Computer Programs v Computer Programs. If you modify a computer program, or make a program using a former program source code, the result must be licensed under GPL. But what about just using or interacting with GPL programs? GPL v2

72 See, GPL v2, section 2.0, paragraph 3.
73 See, GPL v2 section 2.b.
74 See, http://gnu.org/licenses/gpl-faq.html#GPLOtherThanSoftware.
75 Nevertheless, in some particular cases a work can be protected as a computer program and as an artistic work at the same time. That is the border line case of *Algorithm composition* in which, the code is the artistic work. See, https://ccrma.stanford.edu/~blackrse/algorithm.html.

establishes:

> *"These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered* ***independent and separate works in themselves****, then this License, and its terms, do not apply to those sections when you distribute them as separate works"*[76].

Under this provision, If the programs are separate works, interacting is not an issue. But first, the derivative work will have to pass some criteria such as dependency, interaction and location, in order to be considered as independent and separate. Such criteria will be deeply analyzed in the next chapters.

(3) Computer Programs v Program Libraries. Program libraries[77] are small programs, which purpose is to provide functionalities to another program when invoked. Program libraries are everywhere, at the lowest level of the operating system working with the kernel, or at the highest level of a computer program interacting with the user interface.
The GPL license derivative works provisions are not clearly applicable to program libraries. The GPL license has a general perspective, and this fact leads to unclear interpretations of linking program libraries. Program libraries are the core of this investigation, so they will be widely discussed in the next chapters.

1.6. GPL v3 AND DERIVATIVE WORKS

In the derivative works field, the GPL v3 does not change the GPL v2 perspective. Derivative works follow the same independent and particular understanding of what derivative works are, but adding some definitions, and considerations about applicable copyright law.

It defines the term *modify*: *"To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or*

76 See, GPL v2 Section 2.
77 ***"****A program library is simply a file containing compiled code (and data*) that is to be incorporated later into a *program"*. See, Wheeler David, *Program Library HowTo, version 1.36*, United States, 2010. Page 3.

a work "based on" the earlier work"[78].

The copyleft mechanism is established as:

> *"You must license the entire work, as a* ***whole****, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it"*[79].

Derivative works must be licensed under GPL v3 to the whole work, including all parts. But it establishes the possibility of changing such provisions according to the additional terms described in Section 7, and regarding if they are valid under applicable copyright law:

> *"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License,* ***to the extent that they are valid under applicable law****... All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10..."*[80].

(1) Artistic Works v Computer Programs. There is not change. Artistic works are never considered as derivative works[81].

(2) Computer programs v Computer Programs. There is not change. The same perspective already discussed in the GPL v2 license.

(3) Computer programs v Program Libraries. In the specific field of libraries, GPL v3 also establishes that *System libraries are not part of major components*, because they are necessary for the normal functioning of the operating system, so they are not forced to be licensed under the GPL license, or have GPL license compatibility. System libraries which operate with the operating system basic components are not considered as part of the *whole*[82]. However, libraries that are not considered *system libraries*, have

78 See, GPL v3 Section 0. Definitions. Available at: http://www.gnu.org/licenses/gpl.html.
79 See, GPL v3.0, Section 5(c). Conveying Modified.
80 See, Section 7. Available at: http://www.gnu.org/licenses/gpl.html.
81 See, http://gnu.org/licenses/gpl-faq.html#GPLOtherThanSoftware.
82 See GPL v3, Section 1. Source Code. Paragraph 3.

still considered part of the whole:

> *"...For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms* ***that the work is specifically designed to require****, such as by* ***intimate data communication or control flow*** *between those subprograms and other parts of the work"*[83]**.**

This disposition proposes that dynamic shared library source code should be considered as source code of the work when there is a dependency of the work towards the libraries. In contemporary software, a computer program uses many shared libraries. Thus, the library's source code delivery is an important issue to consider.

1.7. THE LINKING EXCEPTIONS

Richard Stallman and the GNU have always been aware about the computer libraries nature and their purposes. That is why they created a particular library generic public license with a weaker copyleft, the LGPL license. However, they still recommend the use of the GPL license for libraries, for developers who prefer strong copyleft on derivative works:

> *"The GNU Project has two principal licenses to use for libraries. One is the GNU Lesser GPL; the other is the ordinary GNU GPL. The choice of license makes a big difference: using the Lesser GPL permits use of the library in proprietary programs; using the ordinary GPL for a library makes it available only for free programs"*[84].

But this provision is not absolute, there is also the system library exception, and in general, there is always the possibility of making exceptions by the copyright holder. There are some historical exceptions to the use of the GPL license that show practical solutions to avoid undesired conflicts.

(1) Lesser GNU General Public License. The GNU project published in the *June 1990 issue of:*

> *"**New library license** We should by now have finished a new alternative General Public License for certain GNU libraries. This license permits linking the libraries into proprietary executables under certain conditions. The new library license actually*

83 See GPL v3, Section 1. Source Code. Paragraph 4.
84 See, http://www.gnu.org/philosophy/why-not-lgpl.html.

> *represents a strategic retreat. We would prefer to insist as much as possible that programs based on GNU software must themselves be free. However, in the case of libraries, we found that insisting they be used only in free software tended to discourage use of the libraries, rather than encourage free* applications"[85].

The GNU *Library General Public License* was created and released in June 1990. This license was the predecessor of the GNU *Lesser General Public License* (LGPL), released in 1999. In the LGPL license, libraries are defined as: *"A library means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables"*[86].

A combined work would not be considered a derivative work under this license, therefore if would be possible to use or modify LGPL libraries into a derivative work with another license:

> *"You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications..."*[87].

But there are some obligations for combined works such as:

> (a) "*Give prominent notice with each copy of the Combined Work that the Library is used...*".
> (b) *"Accompany the Combined Work with a copy of the GNU GPL and this license document"*.
> (c) "*For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library*".
> (d) *"Convey the Minimal Corresponding Source under the terms of this License..."* or *"Use a suitable shared library mechanism for linking with the Library"*.
> (e) "*Provide Installation Information...*"[88].

Despite the fact that the FSF does not recommend the use of the LGPL license, and promotes the use of the GPL license instead[89], a nice solution has been to combine the LGPL v3 license with the GPL v3 license. The LPGL v3 license establishes: *"This version of the GNU Lesser General Public License incorporates the terms and*

85 See, The June 1990 issue, http://free-soft.org/gpl_history/.
86 See, LGPL v3, section 0, paragraph 2.
87 See, LGPL v3, section 4.
88 See, LGPL v3, section 4.
89 See, http://gnu.org/licenses/why-not-lgpl.html.

conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below"[90].

This was a very nice solution because the LGPLv3 includes all the *copyleft* conditions of the GPL v3, but provides flexibility for the specific use of program libraries. The LGPL v3 includes all the GPL v3 license provisions, but adding a more suitable environment for program libraries.

(2) The System Library Exception. GPL v3 establishes: *"The System Libraries of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a* ***Major Component****, but which is not part of that* ***Major Component****..."*[91]. System libraries are not considered part of a *whole*, because they are necessary for the normal functioning of the operating system. System libraries are the essential components of an operating system, and they are not obligated to be licensed under the GPL license, or a GPL compatible license.

(3) The GNU ClassPath Exception. This is a truly GPL linking exception based on the permission of the copyright holder. The GNU ClassPath was a project with the purpose of creating a free software implementation of the standard class library for the Java programming language[92]. This exception consists of a statement distributed within the GPL v2 license:

> "*Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License cover the whole combination. As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module. An independent module is a module which is not derived from or based on this library. If you modify this library, you may extend this exception to your version of the library, but you are not obliged to do so. If you do not wish to do so, delete this exception statement from your version"*[93].

90 See, https://www.gnu.org/licenses/lgpl.html
91 See, GPL v3 section 1. *System libraries* definition.
92 See, http://en.wikipedia.org/wiki/GNU_Classpath.
93 See, http://gnu.org/software/classpath/license.html.

In this case, the copyright holder of the classPath implementation is the FSF itself. The key factor here was the interoperability with the Java Programming language[94]. It was previously said in chapter 1.2(b) that a programming language is not considered subject of copyright, but Java is an Object Object Programming language, category that can test that assumption as we will see later on.

There were other projects that also apply exceptions to GPL like the *gcc Run time library exception*[95], the *Font exception*[96], or the *GNU Guile*[97]. All of them are based on the GNU permission as copyright holder. In the same way, any copyright holder could add a permission and distribute it within the GPL license.

94 It was previously said in chapter 1.2(b) that a programming language is not considered subject of copyright, but Java is an Object Object Programming language, category that can test that assumption as we will see later on.
95 See, http://www.gnu.org/licenses/gcc-exception-3.1.html.
96 See, http://spdx.org/licenses/GPL-2.0-with-font-exception.
97 See, http://www.gnu.org/software/guile/docs/docs-1.6/guile-ref/Guile-License.html.

CHAPTER TWO: TECHNICAL DESCRIPTION OF PROGRAM LIBRARIES LINKING

The purpose of this chapter is to make a short but comprehensive description about *program libraries linking*. But this chapter is not only about program libraries, as it also exposes relevant technical areas which are connected to the field of program libraries such as Object Oriented Programming[98], Computer Memory Address Space[99], Multiple Disk Volumes[100], and Networked[101] Systems. Thus, the structure includes each relevant technical topic in a different sub-chapter.

In order to understand how program libraries work, it is necessary to analyze them under different derivative works criteria, such as: modification, dependency, interaction, time of linking, distribution medium, and location. Understanding the process of linking is essential, because the FSF[102] interpretation of dynamic linking libraries is mostly based on technical facts.

For the readers with basic programming experience, some simple examples will be included. If the reader has not programming experience, this section might still be useful because the important facts of the examples will be analyzed in a *Resume* at the end of each section.

2.1. PROGRAM LIBRARIES

Program libraries can be defined as: *"the contained and compiled code(and data) that*

98 *"programming methodology based on objects, instead of just functions and procedures. These objects are organized into classes, which allow individual objects to be group together"*. See, http://www.techterms.com/definition/oop.

99 *"A computer's address space is the total amount of memory that can be addressed by the computer"*. See, http://www.pcmag.com/encyclopedia/term/37527/address-space.

100 *"In many critical servers, multiple disks are used for performance, reliability, or scalability. The disks are merged and processed so that they look normal but they are not"*. See Page 111, Carrier Brian, *File System Analysis*, Addison Weshley Professional, United States, 2005.

101 *"The purpose of a network is to enable the sharing of files and information between multiple systems"*. See, http://www.techterms.com/definition/network.

102 Free Software Foundation.

is to be incorporated later into a program"[103]. Libraries contain compiled code ready to interact with a program. The linking process is done by a *linker* which binds the libraries to an *executable*[104].

We must understand *contained and compiled code and data*, as code that has been programmed in a programming language such as C[105], and then converted to *object code*[106] with the purpose of linking to an *executable*. To make it more concrete, imagine that a computer program is a material object such as a *Car*. If you want to build a car you need several parts such as an engine, wheels, lights, windows, the coach work, the radio, the GPS[107], and so forth.

The components form together the *Car,* and each component of the car which adds a specific functionality such as the *Wheels*, might be considered a *library*. These functionalities need to link to the car with the purpose of making the car work. We can analyze the *wheel's role* under different criteria:

MODIFICATION: Have the wheels been modified with the purpose of creating the car? Does the car exist without wheels? Or perhaps, Is the union of both which generates the mechanical process?

DEPENDENCY: Are the wheels depending on the car? Is the car dependent on the wheels? Or perhaps, are they interdependent?

INTERACTION: How are the wheels connected to the car? Are they linked by bolts and electric cables? Are they wireless?

TIME OF LINKING: Were the wheels linked to the car at build time? Or perhaps they

---

103 See, Wheeler David, *Program Library HowTo, version 1.36*, United States, 2010. Page 1.

104 *"An executable file is a type of computer file that runs a program when it is opened. This means it executes code or a series of instructions contained in the file"*. See, http://www.techterms.com/definition/executable_file.

105 *"C is a high-level programming language that was developed in the mid-1970s. It was originally used for writing Unix programs, but is now used to write applications for nearly every available platform"*. See, http://www.techterms.com/definition/cplusplus.

106 *"Programming languages such as C and Java are high level languages that require the source code entered by the programmer to be compiled. Once the compiler has processed the code, it produces a set of object code that can be passed to other programs or run on a computer system"*. See, http://www.wisegeek.com/what-is-object-code.htm.

107 Global Positioning System.

link at runtime, when the car turns on.

DISTRIBUTION MEDIUM: Are the wheels sold with the car as a whole? Is it possible to buy the wheels in another store?

LOCATION: Are the wheels at the same physical space than the car?

This *car* example introduces a methodology of analysis based on six relevant criteria that are necessary in order to make a dissection of the GNU derivative work's interpretation in the field of dynamic libraries. These criteria have been considered relevant for the task, and extracted from the GPL v2 license, the GPL v3 license, and the Free Software Foundation interpretations of both[108].

By convention, libraries start with the prefix[109] *lib*. Their suffix[110] is platform dependent. Because the core of this work is based on the GPL license and the GNU, all examples will be provided in a GNU/Linux environment, with little references to other operating systems. There are three prevalent types of *computer program libraries*: Static Libraries, Dynamic or Shared Libraries, and Dynamic Loaded Libraries.

2.2. STATIC LIBRARIES

Static libraries are *collections of object files*[111] that links to the source code of the program, generating an executable at compile time. We can recognize them by the suffix *.a* in Linux, and *.lib* on Windows. Static libraries follow the traditional way of linking libraries within a program, and come from a time when machines had considerable less computing power.

---

108 These criteria will also help to create an *easy to understand* structure for such complex analysis.

109 *"A **prefix** is an affix which is placed before the root of a word"*. See, http://en.wikipedia.org/wiki/Prefix.

110 *"A **suffix** (also sometimes called a **postfix** or **ending**) is an affix which is placed after the stem of a word"*. http://en.wikipedia.org/wiki/Suffix.

111 See, Wheeler David, *Program Library HowTo , version 1.36*, United States, 2010. Page 6.

EXAMPLE

Build a library with two C files: *number1.c* and *number2.c*[112]*:*

number1.c[113]

```
void number1(int *a)
{
  *a=10;                    /* In this function we define that arg a is equal to 10*/
}
```

number2.c

```
void number2(int *b)      /*Here we define that argument b is equal to 5*/
{
  *b=5;
}
```

FIRST: Compile these files into object code, and include them in the static library *libnumbers.a*:

```
~# gcc -c number1.c number2.c              /*Compile source code into object code using gcc[114].*/
~# ar libnumbers.a number1.o number2.o    /*Insert object codes into the  static library using  ar[115].*/
```

SECOND: Create the source code of the program *program.c:*

program.c

```
#include <stdio.h>
void number1(int *);        /*calling function to libnumbers.a(number1.o)*/
void number2(int *);        /*calling function to libnumbers.a(number2.o)*/
main()
{
int a; int b;
number1(&a);
number2(&b);
printf("Number1 is %d , number2 is %d\n", a,b);
return 0;
}
```

112 There are much practical ways to building libraries such as using Cmake, or the GNU libtool. But these non practical examples show a step by step library creating process. Better ways will be described in chapter 2.5.

113 This, and some other examples in this chapter have been inspired from the ones contained in the book Wheeler David, *Program Library HowTo* , version 1.36, 2010. Other Linux tutorials have also influenced these examples, such as: http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html.

114 GNU compiler collection. See, http://gcc.gnu.org/.

115 See, http://linux.about.com/library/cmd/blcmdl1_ar.htm.

THIRD: Generate the binary *executable*, by compiling the program *program.c*, and linking it to the static library *libnumbers.a* :

```
~# gcc -o executable program.c libnumbers.a    /*compile program.c and link it to
                                                  libnumbers.a*/
```

RESUME

MODIFICATION: When *program.c* is linked to *libnumbers.a*, together they generate the binary *executable*. The *executable* is a common descendant of both.

DEPENDENCY: At compile time, *program.c* and *libnumbers.a* become interdependent. They cannot work separately because they are mixed in the binary *executable*.

INTERACTION: *Program.c* interacts with *libnumbers.a* by making function calls such as "*void number1(int *)*". But these function calls are part of the binary *executable*, so they are glued.

TIME OF LINKING: Static library *libnumbers.a* is linked to the program *program.c* at compile time. The result is the binary *executable*.

DISTRIBUTION MEDIUM: The program *program.c* and the library *libnumbers.c* are transformed into the binary *executable*. Therefore they will be distributed as a *whole*.

LOCATION: The library and the program generate the executable. Therefore, they will be located in the same sectors in the storage device, and they will be loaded as a whole into the memory address space when executed.

2.3. DYNAMIC LIBRARIES

Dynamic libraries are also known as *shared* libraries[116]. "*Shared libraries are libraries that are loaded by programs when they start. When a shared library is installed properly, all programs that start afterwards automatically use the new shared library*"[117].

116 For the purposes of this work, they might be called either as Dynamic Libraries or Shared Libraries.
117 See, Wheeler David, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 7.

They have the suffix *.so* in Linux, and *.dll*[118] in Windows systems. Their prefix by convention is also *lib.* Dynamic libraries have advantages in comparison to static libraries. The program is lighter, and the dynamic libraries can be updated separately. Dynamic libraries link to the executable at running time by a linker. The executable binary contents the information about the functions[119] contained in the dynamic libraries required by the program in order to run. There is a dependency list of the dynamic libraries that are meant to link with the executable at runtime.

In GNU/Linux the GNU Linker *ld*[120] is used.

We can easily generate a *dynamic library* with the previous example files *number1.c* and *number2.c*.

EXAMPLE

The same files *number1.c* and *number2.c* will be used. This is a simulation of common program libraries[121].

FIRST: Compile the files number1.c and number2.c into object code. The option *-fPIC* means *enable position independent code*[122]:

```
~#  gcc -Wall  -fPIC  -c numbers1.c numbers2.c        //compile C files   into object code
```

SECOND: Create the shared library *libnumbers.so.1.0*[123] and put our object files into it. The *-shared* option tells the compiler that is a shared library. The *-Wl* option *passes options along the linker*[124]*:*

118 *"Dynamic Link Library." A DLL (.dll) file contains a library of functions and other information that can be accessed by a Windows program."*. See, http://www.techterms.com/definition/dll.

119 In Dynamic linking, these functions remain as undefined symbols until they are linked to the dynamic library. *"After all of the input files have been read and all symbol resolution is complete, the link-editor searches the internal symbol table for any symbol references that have not been bound to symbol definitions. These symbol references are referred to as* ***undefined*** *symbols"*. See, http://docs.oracle.com/cd/E19082-01/819-0690/6n33n7f65/index.html.

120 ***"ld*** *combines a number of object and archive files, relocates their data and ties up symbol references. Usually the last step in compiling a program is to run* ***ld"***. See, http://linux.die.net/man/1/ld.

121 In this example, the Program needs the library in order to function. But it could also be the case that the the Dynamic Library needs the program to function. Such case discuss in the next chapter, when dealing with plug-ins.

122 See, Wheeler David, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 13.

123 This is the **real name** of the library. Don't confuse with the **soname** and the **linker name**.

124 See, Wheeler David, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 13.

```
~# gcc -shared -Wl,-soname,libnumbers.so.1 -o libnumbers.so.1.0  *.o
```

The soname[125] is *libnumbers.so.1* and links to the library real name *libnumbers.so.1.0*[126]. For purposes of updating and selecting convenience, the real name adds a 0 at the end, indicating the library's version number.

Dynamic libraries are usually installed in two special system directories: */lib/* and */usr/lib/*. The reason is that if libraries are shared by several programs, it would be very inconvenient to place them in other directories. However, it is also possible to change the LD_LIBRARY_PATH[127] environment variable, in order to execute the libraries in any directory.

THIRD: Move the new library *libnumbers.so.1* to the library directory:

```
~#  mv libnumbers.so.1.0 /usr/lib          /*move dynamic library into another library directory*/
```

FOURTH: Create symbolic links to the s*oname* and a *linker name*[128]:

```
~# ln -sf /usr/lib/libnumbers.so.1.0 /usr/lib/libnumbers.so.1   /*Create a  symbolic link to the soname
                                                                  with ln[129]*/
~# ln -sf /usr/liblibnumbers.so.1.0 /usr/lib/libnumbers.so      /*Create a symbolic link to  the linker*/
```

FIFTH: Generate the binary *executable* and link it to the Library*:*

```
~# gcc -Wall -L/usr/lib/libnumbers.so.1.0 program.c -lctest -o executable
```

125 *"Every shared library has a special name called the ``soname''. The soname has the prefix ``lib'', the name of the library, the phrase ``.so'', followed by a period and a version number that is incremented whenever the interface changes"*. See, David A. Wheeler, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 8.

126 This name convention is required when using *ELF ld*. It is highly recommended because of other library functions such as indexing and updating. For more info, See, Wheeler David, *Program Library HowTo* , version 1.36, 2010

127 Follow the link instructions in order to set the Library Path: http://www.linuxquestions.org/questions/linux-software-2/how-to-set-ld_library_path-684799/.

128 *"In addition, there's the name that the compiler uses when requesting a library, (I'll call it the ``linker name''), which is simply the soname without any version number"* . See, Wheeler David, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 8.

129 See, http://linux.about.com/od/commands/l/blcmdl1_ln.htm.

When the executable runs, it causes the operating system to load the *dynamic library* into memory:

```
~# ./executable
```

If we want to know the dynamic libraries that are linked to the executable, the simplest way is using the *ldd*[130] command, which shows the shared libraries dependencies:

```
~# ldd  executable

linux-gate.so.1 =>  (0xb7774000)
libnumbers.so.1 => /usr/lib/libnumbers.so.1 (0xb7757000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb75ae000)
/lib/ld-linux.so.2 (0xb7775000)
```

We can note that the *soname* of the dynamic library *libnumbers.so.1.0* is printed. For the purpose of obtaining a deeper analysis of the executable's *ELF*[131] format, and specifically the dynamic section, we can use the command *readelf*[132] :

```
~# readelf -d executable

Dynamic section at offset 0xf20 contains 21 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libnumbers.so.1]
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
 …....
```

This step by step way of generating shared libraries is not practical when building real projects. It is much better to use an automated build system such as *Cmake*[133].

RESUME

MODIFICATION: The binary *executable* does not include the dynamic library

130 *"print shared library dependencies"* . See, http://linux.about.com/library/cmd/blcmdl1_ldd.htm.
131 Executable and Linkable Format. See, http://en.wikipedia.org/wiki/Executable_and_Linkable_Format.
132 *"Displays information about ELF files"*. See, http://linux.about.com/library/cmd/blcmdl1_readelf.htm.
133 See, http://cmake.org.

libnumbers.so.1.0 functions. Thus, there is not a modification of the library by default. The only possible modification occurs at the *memory address space*[134], when running the program.

DEPENDENCY: In technical terms, *libnumbers.so.1.0* and *program.c* are independent. But at running time, our *program.c* becomes dependent on the *libnumber.so.1.0* dynamic library. Nevertheless, the *executable* could also link to other library with similar functionality, and *libnumbers.so.1.0* could also be linked with other programs.

INTERACTION: *Program.c* interacts with *libnumbers.a* by making function calls such as "*void number1(int *)*". These function calls remain separate, and the functions are linked just when the program is invoked, and closed when the process is finished.

TIME OF EXECUTION: Time is the key factor when dealing with dynamic linking. The time of linking is at *runtime*[135]. The *program.c* is linked to *libnumbers.so.1.0* when the *executable* is invoked, and not when the program was compiled.

DISTRIBUTION MEDIUM: Because our library and our program don't form a *whole* in the executable, they can be distributed separately.

LOCATION: The library and the executable are stored in different sectors in the storage device. But at running time, they are combined in the memory address space.

2.4. DYNAMIC LOADED LIBRARIES

Dynamic loaded libraries are dynamic libraries, with the peculiarity that they are linked when loaded, after the startup of a program. This loading/unloading functionality

134 *"An address space is a defined range of locations, physical or virtual, in a memory system. The address may be part of a computer's main memory or storage system, as well as a location within a network host or secondary memory system, such as a graphics card"*. See, http://www.wisegeek.net/what-is-an-address-space.htm.

135 Runtime can also be referred as the Startup time of the Program. See, http://stackoverflow.com/questions/846103/runtime-vs-compile-time.

normally uses an *API*[136], with the purpose of opening the library, looking for symbols, and closing the library[137]. In simple words, they are dynamic libraries with the extra functionality of running just when the functions are invoked, and not necessarily at runtime.

Because of this improvement, dynamic loaded libraries are mostly used with *plug-ins*[138] and *kernel modules*[139]. Their resources and memory management are considerable more efficient than general dynamic linked libraries.

To show this process, the previous dynamic shared library *libnumbers.so.1.0* will be linked to a new program, which will have the option of loading the library functions by an *API*. With the purpose of doing that, in *C* we need to use the <*dlfn.h*> header file, in order to use the dynamic loading *API*[140].

EXAMPLE

FIRST: Create a program for dynamic loading. Four relevant functions are provided within the program: *dlopen()*[141], *dlerror()*[142], *dlsym()*[143], and *dlclose()*[144].

dynamic_loading.c

```
#include <stdio.h>
#include <dlfn.h>

void number1(int *);                    /*calling function to library "libnumbers.so.1.0"*/
void number2(int *);                    /*calling function to library "libnumbers.so.1.0"*/
```

136 Application Programming Interface, See, http://www.techterms.com/definition/api.

137 For more info: See, Wheeler David, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 19.

138 *"A software plug-in is an add-on for a program that adds functionality to it"*. See, http://www.techterms.com/definition/plugin.

139 *"Kernel Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system"*. See, https://wiki.archlinux.org/index.php/Kernel_modules.

140 Other Programming languages also come with default APIs for loading libraries. E.g. In Java, the *ClassLoader* abstract class is responsible for loading classes. See, http://docs.oracle.com/javase/7/docs/api/java/lang/ClassLoader.html.

141 *"opens a library and prepares it for use. In C its prototype is: void * dlopen(const char *filename, int flag); "*. See, Wheeler David, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 20.

142 *"Errors can be reported by calling dlerror(), which returns a string describing the error from the last call to dlopen(), dlsym(), or dlclose()"*. See, David A. Wheeler, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 21.

143 *"The main routine for using a DL library is dlsym(3), which looks up the value of a symbol in a given (opened) library. This function is defined as: void * dlsym(void *handle, char *symbol); "*. See, Wheeler David, *Program Library HowTo* , *version 1.36*, United States, 2010. Page 21.

144 *"The converse of dlopen() is dlclose(), which closes a DL library"*. See, Wheeler David, *Program Library HowTo*, *version 1.36*, United States, 2010. Page 22.

```
main (int argc, char **argv)
{
void *load;
double (*func)(int *);
char *error;
int a; int b;

load=dlopen("'/usr/lib/libnumbers.so.1.0"), RTLD_LAZY);          /*dlopen opens the library. RTLD_LAZY[145] */
if(! Load)
{
 fprintf(stderror, "%s\n", dlerror());
exit(1);
}

func = dlsym(load, "numbers1");                                  /*dlsym  makes possible using the library.Here we
                                                                   call the function numbers1*/
if ((error = dlerror()) != NULL)                                 /*It could also be numbers2, or both*/
{
  fprint(stderror, "%s\n", error);
exit(1);
}
(*func)(&a);
printf("number1=%d\n", a);
dlclose(load);                                          /*dclose() closes a library*/
return 0;
}
```

SECOND: Compile this program:

```
~# gcc -rdynamic  -o executable dynamic_loading.c -ldl
```

As we can see, the functions *number1* or *number2* can be loaded when required by the program.

RESUME

MODIFICATION: The binary *executable* does not include the *libnumbers.so.1.0* components. Thus, there is not a modification of the library by default. The only possible modification is at the c*omputer memory space* when loading the functions of the library.

DEPENDENCY: The library *libnumbers.so.1.0* is independent of the *executable.* There

145 "*In dlopen(), the value of flag must be either RTLD_LAZY, meaning ``resolve undefined symbols as code from the dynamic library is executed'', or RTLD_NOW, meaning ``resolve all undefined symbols before dlopen() returns and fail if this cannot be done''.* For more info see, Wheeler David, *Program Library HowTo , version 1.36,* United States, 2010. Page 20-21.

are two functions inside the library: *number1* and *number2*. When we call any of these functions, the library is linked to the *executable*, but if they are not called, they are not linked. Nevertheless, because one of the typical use of this kind of libraries is in a *plug-in* infrastructure, we should consider that usually plug-ins are programmed for the purpose of enhancing the functionality of a program, so they can be considered program dependent. Thus, a plug-in might need the program, but the program not necessarily.

INTERACTION: *Dynamic_loading.c* interacts with *libnumbers.a* by making function calls such as "*void number1(int *)*". These function calls remain separate, and the functions are binded just when the function is loaded, and closed when the function is unloaded. The function calls are made when requested, so they don't link to the library by default.

TIME OF LINKING: Linking time is loading time. If just a part of the functions included in the library are loaded, it is not precise to assume that all the library is linked as a *whole*.

DISTRIBUTION MEDIUM: Our library and our program are different, so they can be distributed separately.

LOCATION: The library and the executable are stored in different sectors in the storage device. But at running time, they are combined inside the *computer memory address space*, when the functions of the library are loaded.

2.5. COMBINING LIBRARIES AND BUILDING PROJECTS

In a real scenario, static and dynamic libraries are often combined when building a project. This combination might present a real license challenge when choosing between using other libraries or creating new ones. Open source projects are developed by many contributors, and the license compatibility between dynamic libraries might become a

very difficult task[146]. In the world of FOSS[147] Software, several building tools exist towards the project construction. One of the most popular project builder tool combinations are *Make* and *Cmake.*

(1) Make. "*Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files"*[148]. This means that a *Makefile* will build and install a program in one step, but programming a make file can be considered somehow difficult by some users. In practice, *makefiles* are generated automatically by other building tools, such as *Cmake*.

(2) Cmake. *"Is a cross-platform, open source build system*. *CMake is a family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice"*[149].

In our days with Cmake, building programs from its source has become an easy task. We can easily combine static and dynamic libraries to conform a single executable. Cmake generates all needed makefiles in Unix and GNU systems, but it can also build projects for other platforms such as Windows.

EXAMPLE

FIRST: Write a *Cmakelists.txt* file which creates and link static and dynamic libraries:

CmakeLists.txt

```
cmake_minimum_required (VERSION 2.6)
project (numbers)
```

146 If you want to get an idea about the number of shared libraries and licenses that have to co-exist within the project distribution, see https://launchpad.net/ubuntu/quantal/+source/supercollider/+copyright.
147 See, https://en.wikipedia.org/wiki/Free_and_open-source_software.
148 See, http://www.gnu.org/software/make/.
149 For more details see, http://cmake.org.

```
add_library (staticgroup STATIC number1.c number2.c)            //Create a Static library
add_library (dynamicgroup SHARED number3.c number4.c)    //Create a Dynamic Shared Library
add_executable (executable program.c)                              //Generate executable
target_link_libraries(prog staticgroup)                          //Link Static Library to executable
target_link_libraries(prog numbersdynamic)             //Link Dynamic Shared Library to executable
```

With this small *CmakeLists.txt* file we can replace all the command line processes described in the previous chapters. Cmake can create as many libraries as necessary, and link them all to our final executable.

SECOND: We invoke *cmake* and then we run the automatically generated *Makefile*:

```
~# cmake ../
```

THIRD: Run the makefile.

```
~# make
```

Because of *Cmake*, all the necessary flags and options are automatically generated and written into the *Makefile*. The *prefix* and *suffix* of the libraries are generated by default. The *staticgroup library* is renamed to *libstaticgroup.a* and the *dynamicgroup library* is renamed to *libdynamicgroup.so*[150].

RESUME

Building projects are just the practical part of how program libraries are linked with executables in a real scenario. Therefore, the analysis criteria would depend on the type of program libraries that are linked to the executable.

2.6. OBJECT ORIENTED PROGRAMMING

Object Oriented Programming can be defined as "*a programming methodology based on objects, instead of just functions and procedures. These objects are organized into classes, which allow individual objects to be group together*"[151]. Some important

150 In this example, there is not needing of using *soname* or *linker name* for the dynamic library. The update and indexing features are not essential when creating auxiliary extensions, or plug-ins.

151 See, http://techterms.com/definition/oop.

features of OOP languages are i*nheritance*[152]*, and polymorphism*[153].

Regarding the field of linking program libraries, OOP languages have some relevant differences that might change the way that we interpret them. Until now, some simple examples based on the C basics of static and dynamic linking. But now it is time to unleash some of the main differences of OOP languages. In contemporary software, some of the most relevant OOP languages are C++ and Java.

The C++ programming language: *"is a programming language that was built off the C language. The syntax of C++ is nearly identical to C, but it has object-oriented features, which allow the programmer to create objects within the code"*[154]**.** It comes from 1983[155], and it is widely used in all kinds of applications. The success of c++ is bound to the possibility of using C programs with an object oriented programming orientation. In the following example we can get the flavor of a c++ program[156]:

Polygons.cpp

```
#include <iostream>
using namespace std;

class Polygons {                          //main class
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
  };
```

152 *"In object-oriented programming (OOP),* ***inheritance*** *is a way to establish Is-a relationships between objects. In classical inheritance where objects are defined by classes, classes can inherit attributes and behavior from pre-existing classes called baseclasses, superclasses, or parent classes. The resulting classes are known as derived classes, subclasses, or child classes".* https://en.wikipedia.org/wiki/Inheritance_%28object-oriented_programming%29. For a complete inheritance description, see: Eckel Bruce, Thinking in Java 4th edition, page 21- 23.

153 *"Subtype polymorphism, often referred to as simply* ***polymorphism*** *in the context of object-oriented programming, is the ability to create a variable, a function, or an object that has more than one form".* See, *https://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming*. For a complete polymorphism description, see: Eckel Bruce, Thinking in Java 4th edition, page 21- 23.

154 See, http://www.techterms.com/definition/cplusplus.

155 C++ was developed by Bjarne Stroustup since 1979 at Bell Labs. See, https://en.wikipedia.org/wiki/C%2B%2B.

156 A highly recommended C++ introductory book is: Eckel Bruce, *Thinking in C++ vol1*, New Jersey, Prentice Hall Inc, 2000.

```
class Rectangle: public Polygons {          //Rectangle is a derived class of Polygons
  public:
    int area ()
      { return (width * height); }
  };

class Triangle: public Polygons {           //Triangle is a derived class of Polygons
  public:
    int area ()
      { return (width * height / 2); }
  };

int main () {
  Rectangle rectangle;
  Triangle triangle;

  rectangle.set_values(2,3);                //The rectangle class uses the Polygon's
  triangle.set_values(2,3);
  cout << rectangle.area() << endl;
  cout << triangle.area() << endl;

  return 0;
}
```

In this simple example, the object main class is *Polygons*, and the derived classes are *Rectangle* and *Triangle*. The method[157] *.set_values(2,3)* calls the *set_values(int a, int b)* function of the *Polygons* class, and has the task of setting the values of the arguments *int a*, and *int b*. *Rectangle* and *Triangle* are derived from *Polygons*, therefore they inherit the *Polygons* class methods. We can always create as many derived classes as we need of the *Polygons class*, and of the *Rectangle* and *Triangle classes*.

The Java programming language: "*Java is a programming language and computing platform first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!"*[158]. Java

157 "*In object oriented programming, a **method** is a subroutine (or procedure) associated with a class. Methods define the behavior to be exhibited by instances of the associated class at program run time*". See, http://en.wikipedia.org/wiki/Method_%28computer_programming%29.

158 See, http://www.java.com/en/download/faq/whatis_java.xml.

brought a new programming environment which has some important differences with his predecessors[159]. Java has two relevant characteristics which are different from other OOP languages:

(1) Is platform and device independent. There is not a compilation of executable code for a particular machine. Instead, Java uses the *Java byte code*[160], a special format which can be understood in all platforms. This *byte code format* contains the instructions of a Java program, which will be executed by the *Java virtual machine*[161] . In simple words, *Java byte code* is platform independent, as it can be understood by all operating systems and even hardware devices[162].

(2) All libraries are dynamically linked in Java. The *Java virtual machine* executes the *Java byte code* instructions at run time[163]. The JVM contains the standard Java libraries[164], and they are dynamic loaded libraries. This means that instead of *link a complete program before execution*, the classes and interfaces[165] are linked and loaded during the execution of the program[166].

EXAMPLE

FIRST: This example shows most Object Oriented Programming features in Java:

159 Such as C++. or Smalltalk.

160 *"**Java bytecode** is the form of instructions that the Java virtual machine executes"*. See, http://en.wikipedia.org/wiki/Java_bytecode.

161 *"is a virtual machine that can execute Java byte code. It is the code execution component of the Java platform"*. See, http://en.wikipedia.org/wiki/Java_virtual_machine.

162 However, *"only the interpreter and a few native libraries need to be ported to get Java to run on a new computer or operating system"*. See, http://wiki.answers.com/Q/Why_java_is_platform_independent.

163 Nevertheless, there are mechanisms for including files to run in a specific machine. See, http://www.ehow.com/info_12216358_java-static-linking.html.

164 The Java Class Library extends the programmer options. *"The **Java Class Library** (JCL) is a set of dynamically loadable libraries that Java applications can call at run time"*. See, http://en.wikipedia.org/wiki/Java_Class_Library#Licensing

165 *"An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements"*. See, http://www.tutorialspoint.com/java/java_interfaces.htm.

166 See, Drossopoulou, Eisenbach, *Manifestations of Java Dynamic Linking – an approximate understanding at source language level*-, Imperial College, London, 2002.

polygons.java

```
import java.util.*;
 import java.math.*;

 interface GetPolygons                          //We use an Interface
{
            public float width = 4;
            public float height = 3;
            public void get();
            }
 class GetRectangle implements GetPolygons    //class GetRectangle implements Getpolygons
{
            public void get()
            {
              System.out.println("Rectangle is :" + (4 * 3));
              }
            }

 class GetTriangle implements GetPolygons      //class GetTriangle implements GetPolygons
{
            public void get()
            {
              System.out.println("Triangle is :" + (4*3)/2);
              }
            }

 abstract class Formula                         /* This abstract class[167]  is included with the purpose
                                                   of showing polymorphism features */
{
            private GetPolygons getPolygons;
            public Formula()
            {
            }
            public void setGetPolygons (GetPolygons polygons)
            {
                        getPolygons = polygons;
                        }

            public void get(){
            getPolygons.get();
            }
}

 class Rectangle extends Formula              //class Rectangle is derived from Formula
{
  public Rectangle()
  {
  setGetPolygons(new GetRectangle());
  }
}

 class Triangle extends Formula                //class Triangle is derived from Formula
{
  public Triangle()
  {
```

167 *"An abstract class is a class that is declared abstract it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed".* See, http://docs.oracle.com/javase/tutorial/java/IandI/abstract.html.

```
  setGetPolygons(new GetTriangle());
 }
}


class GetThemAll                    //This class will get executed by the Java virtual machine
{
        public static void main (String[] args)
        {

    Rectangle rectangle = new Rectangle();
       Triangle triangle = new Triangle();

       rectangle.get();
       triangle.get();

        }
       }
```

SECOND: Use the Java compiler *javac*[168], in other to obtain the *java byte code* of all *classes:*

```
~# javac   polygons.java
```

THIRD: Execute the *Java byte code* of the class *GetThemAll*, and all classes get linked at run time:

```
~# java  GetThemAll

Rectangle is : 12
Triangle is : 6
```

As we have seen, Object Oriented Programming presents new challenges because of their inheritance nature. Also consider that the process of linking libraries is different in the Java language.

RESUME

168 *"**javac** (pronounced "java-see", or often "javack") is the primary Java compiler, included in the Java development Kit (JDK) from Oracle Corporation.* See, http://en.wikipedia.org/wiki/Javac.

MODIFICATION: There is an inherited condition, *former classes* are modified into *derived classes,* with the purpose of extending their functionality.

DEPENDENCY: *Derived classes* are dependent on f*ormer classes*.

INTERACTION: *Derived classes* interact with *former classes* by using the *former class methods.*

TIME OF EXECUTION: Depends if the libraries are static or dynamic. In the case of the Java language, libraries are dynamically loaded.

DISTRIBUTION MEDIUM: Derived classes and former classes could be distributed separately. But that is not usual, considering that there are official distributions either for the proprietary world, or the FOSS world.

LOCATION: They might be stored in different sectors in the storage device. But when the derived classes are loaded into memory[169], the methods and variables of the former class will be instantly loaded as well, so they would share the same location in the *computer memory address space.*

2.7. NETWORK SOFTWARE

In our days, network software systems are very common. Software can be interconnected because of special types of software components called *pipes*[170] and *sockets*[171]. The difference between both is that *pipes* send unidirectional communication, while *sockets* are bi-directional, therefore, more suitable for network communication. Sockets functions are often included in standard libraries such as the <*sys/socket.h*>

169 They could be loaded into memory at run time(if shared libraries), or at the loading time (if dynamic loaded libraries).
170 *"A pipe is a communication device that permits unidirectional communication"*. See, Mitchell, Oldham, Samuel, *Advanced Linux Programming,* New Riders Publishing, United States, 2001. Page 110.
171 *"A socket is a bidirectional communication device that can be used to communicate with another process on the same machine or with a process running on other machines"*. See, See, Mitchell, Oldham, Samuel, *Advanced Linux Programming,* New Riders Publishing, United States, 2001. Page 116.

header file contained in the *GNU C Library*[172], the *C++ Sockets Library*[173], or the *Java Sockets Library*[174].

EXAMPLE

A well known GPL licensed computer program for network analysis is *Wireshark*[175], and it uses the default X Windows system for GNU/Linux named *Xorg*[176]. They are different processes which are communicated by *shared memory segments*[177].

FIRST: Check the *shared memory segments* with the *ipcs*[178] command*:*

```
~# ipcs -m
0x00000000 229381     root      600        393216     2          dest
0x00000000 262150     root      600        393216     2          dest
…
```

SECOND: Check the *processes* which are connected to the *memory segment ID* 229381 with *lsof*[179] and *grep*[180]:

```
~# lsof | grep 229381
Xorg        1102      root  DEL       REG        0,4        229381 /SYSV00000000
wireshark   3518      root  DEL       REG        0,4        229381 /SYSV00000000
….
```

When two programs are interconnected, each one has a different *process*[181], but they are connected by the shared memory segments. Shared libraries once loaded, can be mapped

---

172 *"Any Unix-like operating system needs a C library: the library which defines the ``system calls'' and other basic facilities such as open, malloc, printf, exit..."*. See, http://www.gnu.org/software/libc/.

173 See, http://www.trumphurst.com/cpplibs2.html.

174 See, http://docs.oracle.com/javase/1.5.0/docs/api/java/net/Socket.html.

175 *"Wireshark is the world's foremost network protocol analyzer. It lets you capture and interactively browse the traffic running on a computer network"*. See, http://www.wireshark.org/about.html.

176 *"Xorg is the public, open-source implementation of the X window system version 11"*. See, https://wiki.archlinux.org/index.php/Xorg.

177 A very comprehensive post about shared memory segments can be found at: http://www.orafaq.com/node/8.

178 *"Shared Memory is an efficeint means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access"*. See, http://www.cs.cf.ac.uk/Dave/C/node27.html.

179 *"list open files"* . See, http://linux.about.com/library/cmd/blcmdl8_lsof.htm.

180 *"searches one or more input files for lines containing a match to a specified pattern"*. See, http://www.gnu.org/software/grep/.

181 *"A running instance of a program is called a process"*. See, Mitchell, Oldham, Samuel, *Advanced Linux Programming*, New Riders Publishing, 2001. Page 45.

into different *processes.* This is possible using a *copy-on-right*[182] technique. Thus, the shared library instead of been copied many times, is just pointing to the new processes.

In the network, the connection is done via sockets, because of their bidirectional communication properties. Contemporary software is based on two models of communication: The Client Server model, and the Peer to Peer model.

(1) The Client Server model. This model is the most widely used in inter-process communications. In simple terms, there are two processes running. The client process requests the connection to the server process. In networking the situation does not change, the client host requests the connection to the server host, and the server host accepts or denies the connection.
For example, when using the *HTTP*[183] protocol on the Internet, if you connect from your computer to a *Web Site*, you are the client, and your requested *Web Site* is the Server. Very common connection oriented programs such as *Netcat*[184], or *Open SSH*[185], follow this model. In technical terms, this connection is possible because of the *Client Host* and *Server Host* sockets.

(2) The Peer to Peer Model. In this model all peers are expected to be server and client at the same time. The sockets of a host connect to the sockets of other host in both directions, as a requester and as a listener. For example, when using file sharing software such as *Bit Torrent*[186]. All hosts running a *Bit Torrent client* are able to

182 *"**Copy-on-write** stems from the understanding that when multiple separate tasks use identical copies of the same information (i.e., data stored in computer memory or disk storage), it is not necessary to create separate copies of that information for each process, instead they can all be given pointers to the same resource"*. See, http://en.wikipedia.org/wiki/Copy-on-write.

183 *"Stands for "HyperText Transfer Protocol." This is the protocol used to transfer data over the World Wide Web"*. See, http://www.techterms.com/definition/http.

184 *"Netcat is a featured networking utility which reads and writes data across network connections, using the TCP/IP protocol"*. See, http://netcat.sourceforge.net/.

185 *"OpenSSH encrypts all traffic (including passwords) to effectively eliminate eavesdropping, connection hijacking, and other attacks. Additionally, OpenSSH provides secure tunneling capabilities and several authentication methods, and supports all SSH protocol versions"*. See, http://www.openssh.org/.

186 **"** *is a protocol that supports the practice of peer to peer file sharing and is used for distributing large amounts of data over the Internet"*. See, http://en.wikipedia.org/wiki/BitTorrent.

download files from other hosts, and seed other hosts with their own files. All these services are coordinated by a *Bit Torrent tracker*, which helps the hosts to synchronize between each other, with the purpose of exchanging files. All hosts are connected using sockets.

This model has become very popular in the last ten years. Very popular P2P software is *Skype*[187], or *Tor*[188].

RESUME

MODIFICATION: Generally, there is not modification because the client's process and the server's process have different *executables*. Nevertheless, if the dynamic library of a host is linked to another host, and there is only one process running, the context is different. This might be the case of a program linking to a library in the network.

DEPENDENCY: In the client server model, the Client host depends on the Server host. In the P2P model, all the peers are interdependent. Depends on who is seeding and who is receiving the packets.

INTERACTION: They interact their processes with their sockets, pipes or via shared memory segments.

TIME OF EXECUTION: The client and server programs are running on different hosts. The libraries and executables of both are normally linked separately.

DISTRIBUTION MEDIUM: Programs and libraries can be distributed separately.

LOCATION: The Hosts might be located in different places, at different computer networks. But when the programs are connected, they generate different *processes*, therefore, they are allocated at different places if they are in different volumes..

187 *"The service allows users to communicate with peers by voice using a microphone, video by using a webcam, and instant messaging over the Internet"*. See, http://en.wikipedia.org/wiki/Skype.

188 *"Tor is free software and an open network that helps you defend against a form of network surveillance that threatens personal freedom and privacy, confidential business activities and relationships, and state security known as traffic analysis"*. See, https://www.torproject.org/.

2.8. COMPUTER MEMORY SPACE

The Memory is a physical group of RAM chips. "*It determines the size and number of programs that can be run at the same time, as well as the amount of data that can be processed instantly*"[189]. This means that all programs when loaded, generate a process in the memory space.

There are two kinds of computer memory:

(1) RAM Memory. "*(Random Access Memory) The main memory in a computer, smartphone or tablet. It is used as a temporary workspace to execute instructions that process the data*"[190]. Normally physical memory and RAM memory are referred as the same. When the programs are loaded, they generate a process which allocates memory physical space in the RAM. It can be *Dynamic*[191] or *Static*[192].

(2) Virtual Memory. "*Virtual memory allows multiple programs to load in memory at the same time, virtual memory uses the hard disk to temporarily hold what was in real memory*"[193]. Virtual memory is often referred as swap or cache memory. It optimizes memory management because it uses the hard drive space to record the memory contents into page segments[194]. In Linux based systems, it is often recommended using a special partition called the *swap partition*[195], in order to manage the hard drive space for the virtual memory.

Processes run independently of each other, so when an executable is loaded, it generates a process which is dynamically allocated into a virtual memory system, allowing the

189 See, http://www.pcmag.com/encyclopedia_term/0,1237,t=memory&i=46756,00.asp.
190 See, http://www.pcmag.com/encyclopedia_term/0,1237,t=RAM&i=50159,00.asp.
191 "*The most common type of computer memory. Dynamic RAM (DRAM, D-RAM) chips are very dense because they use only one transistor and one storage capacitor for each bit*". Copied from http://www.pcmag.com/encyclopedia/term/42192/dynamic-ram.
192 "*A fast memory technology that requires power to hold its content. Static RAM (SRAM, S-RAM) is used for high-speed registers, caches and relatively small memory banks such as a frame buffer on a display adapter*". Copied from http://www.pcmag.com/encyclopedia/term/52041/static-ram.
193 See, http://www.pcmag.com/encyclopedia_term/0,1237,t=virtual+memory&i=53929,00.asp.
194 Generally, Segments of 4kb
195 See, http://www.linux.org/article/view/swap-partition.

separation of the processes. The virtual address space contains blocks of memory addresses. Typically, memory addresses are 32-bit mode or 64-bit mode. These virtual addresses are mapped into physical memory by the page segments[196].

EXAMPLE

When the program links to the dynamic libraries, they share the same computer process in the computer memory address space. For example, when using a program such as the text editor *Gedit*[197], Gedit links to dynamic shared libraries such as *libXau.so.6.0.0*[198], *libdbus-1.so.3.5.8*[199], and so forth. When Gedit is executed, it links to the dynamic libraries, and they form a single *process*[200]:

FIRST: Run the gedit executable.

```
~# ./gedit                    //run the gedit executable
```

SECOND: Find dynamic libraries dependencies from the ELF[201] binary format with readelf:

```
~# readelf -d gedit

0x00000001 (NEEDED)          Shared library: [libgedit-private.so.0]
 0x00000001 (NEEDED)         Shared library: [libX11.so.6]
….....................
```

THIRD: Find open files with lsof[202], and filter the Gedit process within the shared libraries with grep[203].

196 For a detailed and easy description of Memory Allocation, See this blog: http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory.
197 *"gedit is the official text editor of the GNOME desktop environment"*. See, http://projects.gnome.org/gedit/.
198 See, http://www.linuxfromscratch.org/blfs/view/svn/x/libXau.html.
199 See, http://dbus-cplusplus.sourceforge.net/.
200 *"A process is an executing (i.e., running) instance of a program. Each process is guaranteed a unique PID, which is always a non-negative integer"*. See, http://www.linfo.org/pid.html.
201 The *Executable and Link format* is in charge of doing the Linking between the Executable and the dynamic libraries. See, http://www.linux-mips.org/wiki/ELF.
202 For more information about Open Files and processes, See, http://www.ibm.com/developerworks/aix/library/au-lsof.html.
203 *"searches one or more input files for lines containing a match to a specified pattern"*. See,

```
~# lsof | grep gedit

gedit    2639 luisenriquez  mem       REG       8,4   774568 16911158 /usr/lib/libgedit-private.so.0.0.0
gedit    2639 luisenriquez  mem       REG       8,4  1254264 16912828 /usr/lib/i386-linux-gnu/libX11.so.6.3.0
….....................
```

As we can see, both dynamic linked libraries share the **2639** process within the Gedit process.

At the physical memory level, the *ELF*[204] binary format is in charge of linking the shared libraries. The bits allocated in the *process* can also can be found inside the *data structures*[205] of the computer physical memory. The *data structures* in the memory are a *bit stream*[206] representation of all processes loaded into the memory.

When dynamic libraries are loaded into the memory, they are mapped to all the processes which require them. *The process* is temporary, and it will be deleted as soon as the device turns off. Nevertheless, a *bit stream image*[207] of the physical memory can help as proof of the existence of any computer *process* that was loaded at a certain time. In a *bit stream* image is possible to find the bits corresponding to ELF formats, in order to establish dynamic linking between shared libraries and executables.

In Unix environments, the *dd*[208] command can be used for obtaining a *bit-stream* copy of the memory:

```
~$ dd if=/dev/fmem of=/home/ram.dd bs=512 count=4194304    /*Copy the first 4194304 sectors of the
                                                            physical memory*/
```

The ELF file data structures and the shared libraries can be found in the *bit stream*

http://www.gnu.org/software/grep/.

204 The *Executable and Link format* is in charge of doing the Linking between the Executable and the dynamic libraries. See, http://www.linux-mips.org/wiki/ELF.

205 "*Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address—a bit string that can be itself stored in memory and manipulated by the program*". See, http://en.wikipedia.org/wiki/Data_structure.

206 "***bit stream** is a time series or sequence of bits. A bytestream is a series of bytes, typically of 8 bits each, and can be regarded as a special case of a bitstream*". See, https://en.wikipedia.org/wiki/Bitstream.

207 "*A bit-stream image is a sector-by-sector / bit-by-bit copy of a hard drive*". See, http://www.computer-forensics.net/FAQs/what-is-a-bit-stream-image.html

208 "*convert and copy a file*". See, http://linux.about.com/od/commands/l/blcmdl1_dd.htm.

image, with the aid of a hexadecimal editor such as xxd[209]:

```
~# dd if=/ram.dd  skip=1516466 count=1 | xxd

0000020: 0000 0000 0100 0000 6c69 6267 6564 6974  ........libgedit
0000030: 2d70 7269 7661 7465 2e73 6f2e 302e 302e  -private.so.0.0.
0000040: 3000 0000 0000 0000 0000 0000 0000 0000  0...............
```

```
~# dd if=/ram.dd  skip=1517293 count=1 | xxd

0000020: 0000 0000 0100 0000 6c69 6258 3131 2e73  ........libX11.s
0000030: 6f2e 362e 332e 3000 0000 0000 0000 0000  o.6.3.0.........
```

```
~# dd if=/ram.dd  skip=1810688 count=1 | xxd

0000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............
0000010: 0300 0300 0100 0000 14e4 ffff 3400 0000  ............4... ....
0000020: 9404 0000 0000 0000 3400 2000 0400 2800  ........4. ...(. ...
```

RESUME

The running process is temporal, and it will be deleted as soon as the process is killed or the computer is turned off. The *computer memory address space* is the place where the executable links to the program libraries. It is possible to obtain a copy of the physical memory with the purpose of proving that program libraries and executables shared *data structures* at the physical memory level.

2.9. VOLUMES, PARTITIONS AND MULTIDISKS

Very often, *hard drive* and *volume* are referred as synonyms, but in fact they mean different things. A *hard drive* is a storage medium with a certain capacity measured in *bytes*, which are divided into *sectors*[210]. A *volume* is a single logical storage area within a single file system. At the volume level, the information is stored is *blocks*[211] or *clusters*[212]. *blocks* and *clusters* are groups of sectors.

209 *"xxd creates a hex dump of a given file or standard input"*. See, http://linux.about.com/library/cmd/blcmdl1_xxd.htm

210 *"is the smallest addressable storage unit in the hard disk and is typically 512 bytes. Each sector is given an address, starting at 1 for each track"*. See, Brian Carrier, *File system analysis*, Addison Wesley Professional, United States, 2005. Page 31.

211 *"The Unix communities employ the term block to refer to a sector or group of sectors"*. See, http://en.wikipedia.org/wiki/Cylinder-head-sector.

212 *"Clusters are allocation units for data on various* ***windows*** *file systems (FAT, NTFS, etc.)"*. See,

It is possible to have many volumes on a hard drive, and it is also possible to have many hard drives in a volume. We have many *volumes in a hard drive*, when the hard drive has different partitions on it. We could have a partition running *Windows* with a *NTFS*[213] file system, a second one running *Ubuntu*, with *Ext3*[214] file system, a third one with a *swap partition*[215], a fourth one running *Mac OSX* with a *hfsplus*[216] file system, and so forth. Every partition is a *volume*.

We could also have many *hard drives in a volume*, when the partition uses several devices, for different reasons such as improving performance and storage capacity. Very common multi-disk volumes are *RAIDS*[217] and *Disk Spanning*[218].

RESUME

An *executable* links to *dynamic libraries* and form a process in the c*omputer memory space*. But these components normally need to be stored in the same *volume*. If they are located in different *volumes*, a Networking system must be applied, such as the *client - server* model, or the *P2P* model.

---

http://en.wikipedia.org/wiki/Cylinder-head-sector.

213 *"The New Technologies File System (NTFS) was designed by Microsoft and is the default file system for Microsoft Windows"* . See, Brian Carrier, *File system analysis*, Addison Wesley Professional, United States, 2005.

214 *"are the default file systems for many distributions of the Linux operating system"*. See, Brian Carrier, *File system analysis*, Addison Wesley Professional, United States, 2005.

215 See, http://www.linux.org/article/view/swap-partition.

216 HFS Plus or HFS+ is a file system developed by Apple Inc. to replace their Hierarchical File System (HFS) as the primary file system used in Macintosh computers (or other systems running MacOS).

217 *"RAID stands for Redundant Arrays of Inexpensive Disks and is commonly used in high-performance systems or in critical systems"*. See, Carrier Brian, *File system analysis*, Addison Wesley Professional, United States, 2005. Pages 111-116.

218 *"Disk spanning makes multiple disks appear to be one large disk"*. See, Carrier Brian, *File system analysis*, Addison Wesley Professional, United States, 2005. Pages 117-120.

CHAPTER THREE: THE GPL FAQ INTERPRETATION OF LINKED LIBRARIES

The field of linked libraries is still an obscure and not well understood GPL area. The GPL license goes far beyond the provisions of traditional copyright law, and this challenge is very positive, because is contributing to the evolution of copyright law. Due to the complexity of the subject, the FSF interprets the GPL license with the purpose of helping developers, lawyers, and users, to understand the meaning of all GPL provisions.

However, it is certainly difficult to interpret a generic-purpose license just in one direction, and especially the GPL, because of the *copyleft* viral effect. A generic-purpose license is a copyright license, with the difficult task of being interpreted by different *applicable laws*, in different *jurisdictions*, and different *law families*[219].

The FSF interpretations are contained in the GPL FAQ[220]. They should be understood as the interpretations of the *copyright license* writer, who provides the option of using the GPL license to the copyright holders. However, this interpretation must also be understood inside the boundaries of copyright national laws, and international copyright conventions.

An important issue to consider, is that the GPL FAQ makes a distinction between static and dynamic libraries in some of their interpretations. They expose some different technical situations regarding the particular conditions of both. For that reason, the type of linking becomes relevant in this chapter, even if such distinction does not exist in national copyright laws, or international copyright conventions.

219 For example, consider that in the USA, a copyright license is also considered a contract. See, http://www.law.washington.edu/lta/swp/law/contractvlicense.html.
220 See, http://www.gnu.org/licenses/gpl-faq.html.

The purpose of this chapter is to analyze the GPL FAQ interpretations of linking program libraries and derivative works, following five criteria[221] from the previous chapter: modification, dependency, interaction, distribution medium, and location. These criteria is very valuable because it splits the GPL license copyleft restrictions into specific categories, and the confrontation of such categories inside a copyright law context. This confrontation creates several paradigms.

3.1. MODIFICATION

The Modification criterion in the GPL license determines if the original works have been modified or copied into another work.

GPL v2: *"You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program..."*[222]. Modification is a general requirement for creating a derivative work.

A cause condition is established: *"You must cause any work that you distribute or publish, that* ***in whole or in part*** *contains or is derived from the Program or any part thereof, to be licensed as a whole..."*[223].

However, identifiable sections of a work which are not derived works would not have to follow the license: *"...If* ***identifiable sections*** *of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works"*[224].

GPL v3: *"To "modify" a work means* ***to copy from or adapt all or part*** *of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on"*

221 The *time of execution criterion* was useful in order to make a difference between *static libraries*, *dynamic linked libraries* and *dynamic loaded libraries*. However, the analysis of the other five criteria will be directly applied to the GPL FAQ interpretations, assuming such *time of execution* differences.

222 See, GPL v2 section2 paragraph 1. Available at http://www.gnu.org/licenses/gpl-2.0.html.

223 See, GPL v2, section 2, paragraph 1.b, available at, http://www.gnu.org/licenses/gpl-2.0.html.

224 See, GPL v2, section 2, paragraph 2, available at http://www.gnu.org/licenses/gpl-2.0.html.

the earlier work[225]*"*.

Under this *modify* definition, *to copy from* also generates a *modified version*, if it is not an *exact copy*.

GPL FAQ:

"If the modules are included in the same executable file, they are definitely combined in one program. If modules are designed to run linked together in a ***shared address space****, that almost surely means combining them into one program"*[226].

The dynamic libraries and the executable are combined into a single process. They also share the same memory address space.

In respect of Object Oriented Programming: "***Subclassing is creating a derivative work***. *Therefore, the terms of the GPL affect the whole program where you create a subclass of a GPL'ed class"*[227]. This interpretation covers all the object oriented programming cases.

ANALYSIS

The modification criterion can be applied under two scenarios: *The process*[228] *as a modified work* of the dynamic libraries, and the o*bject oriented programming* as an exception to the general programming language perspective.

(1) The Process as modified work. In static linked libraries, the modified work is clearly the *executable*, because it contains the source code, and the static library components. In dynamic linked libraries, the executable does not contain the library functions, just an expectation to be linked. Until then, these instructions remain as *undefined symbols*. Therefore, the equivalent will be a *process* running at the computer

225 See, GPL v3 section 0.
226 See, http://www.gnu.org/licenses/gpl-faq.html#MereAggregation.
227 See, http://http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#TOCOOPLang.
228 *"A process is an executing (i.e., running) instance of a program. Each process is guaranteed a unique PID, which is always a non-negative integer"*. See, http://www.linfo.org/pid.html.

memory address space[229]. This process is temporary, and it will disappear as soon as the device is turned off[230].

The *process* is a running instance of a program. If the *process* is considered a modified work, a very controversial paradigm emerges: *Who creates the modified work?* The answer is, the *user*. Then, if the *user* does not distribute the modified version, and uses the program for private purposes, there wouldn't be a copyleft infringement, or at least under this criterion. The GNU disposes: *"The GPL does not require you to release your modified version, or any part of it. You are free to make modifications and use them privately, without ever releasing them"*[231]. This means that the *user* would not be the infringer, because he is allowed to make modifications and run them privately. If the *user* redistributes the program and the libraries, they won't be linked, so the cycle repeats and repeats.

Nevertheless, the *developer* who violates the GPL provisions in this way could also be considered as an indirect responsible of the copyright infringement. In some jurisdictions such as the United States, *secondary liability*[232] might apply because of the the contribution and facilitation for committing infringement[233]. The panorama is still unclear, and there are not legal precedents of secondary liability in the field of linking libraries and the GPL license.

Finally, we should question if the process is a derivative work, or is the program itself. Technically, the process is an instance of the program. Legally, there is not distinction between the program and the process. The process is the instance where the executable gets completed. Therefore, the *executable* of the program linked to dynamic libraries, only exists as a temporary process.

229 This temporary location of the process will be analyzed in the *location* criteria, in chapter 3.5.
230 All details were already described in the Chapter 2.8 of this work.
231 See, http://www.gnu.org/licenses/gpl-faq.html#GPLRequireSourcePostedPublic.
232 This facilitation for others infringement is also known as *Contributory liability*. A well known case is *Gershwin Publishing Corp. v Columbia Artists Management*, Inc. 443_F.2d_1159(2d Cir. 1971).
233 Liability is generally determined by national laws.

(2) Object Oriented Programming. The FSF interpretation about OOP languages is straightforward, sub-classing is not allowed. A new paradigm emerges when we consider that all source files in some OOP languages are derived from one *object class*[234]. Then, all programs would have to follow the programming language license.
In terms of the Berne Convention, if programming languages don't have an *expression*, they might not be subject of copyright protection. In addition, *class methods* could also be part of Application Programming Interfaces. It is not clear yet if using APIs is or not considered as *fair use* exception[235].

3.2. DEPENDENCY

The *dependency* criterion on the GPL license determines if the libraries are dependent on the program, or the program depends on the libraries.

GPL v2: *"If identifiable sections of that work ... are not derived from the Program, and can be **reasonably considered independent**... Then this License, and its terms, do not apply to those sections when you distribute them as separate works"*[236].

GPL V3: *"A compilation of a covered work with other separate and **independent** works ... is called an aggregate"*[237].

But, *What is considerably independent?* The license does not provide such definition.

GPL FAQ: The interpretation provides a dependency criterion: "*If your program depends on a non-free library to do a certain job, it cannot do that job in the Free World. **If it depends on a non-free library** to run at all, it cannot be part of a free*

234 In most OOP Languages such as *Super Collider*, all *classes* are derived from one *object class*. See, http://supercollider.sourceforge.net/.
235 An important legal precedent which involves *the GPL license*, *application programming interfaces*, and *fair use* is *Oracle v Google*. This case will be deeply analyzed in chapter 4.3 of this work.
236 See, GPL v2 section 2, paragraph 2.
237 See, GPL v3 section 5, paragraph 2. Available at http://www.gnu.org/licenses/gpl.html.

operating system such as GNU; it is entirely off limits to the Free World"[238]. This provision certainly determines that in case of any dependency, the GPL license is not negotiable. It does not matter if it is part of a certain job, or about all the program functionality.

If the libraries are GPL, the interpretation is the same: **Q:***"If a library is released under the GPL (not the LGPL), does that mean that any software which uses it has to be under the GPL or a GPL-compatible license?"* **A:***"Yes, because the software as it is actually run includes the library"*[239].

ANALYSIS

The dependency criterion can be applied over two scenarios:

(1) When a GPL Program depends on a non-free Library. When a new software project is built, it becomes crucial to determine which libraries are useful for providing the required functionalities. When non free libraries are needed, the copyright holder of those libraries can still give his permission to use them. In such case, there is not copyright infringement.

If those libraries are used by the copyright holder of a GPL licensed program, a new paradigm emerges: The infringement comes from the copyright holder who uses these non-free libraries for his GPL licensed program. That is not possible under most copyright laws because he already got the permission for using them. For avoiding such cases, the GPL FAQ adds the following:

> *"If the program is already written using the non-free library,* ***perhaps it is too late to change the decision.*** *You may as well release the program as it stands, rather than not release it. But please mention in the README that the need for the non-free library is a drawback, and suggest the task of changing the program so that it does the same job without the non-free library"*[240].

This panorama gets more complicated if we consider that contemporary software

238 See, http://www.gnu.org/licenses/gpl-faq.html#FSWithNFLibs. Paragraph 1.
239 See, http://www.gnu.org/licenses/gpl-faq.html#IfLibraryIsGPL.
240 Copied from: http://www.gnu.org/licenses/gpl-faq.html#FSWithNFLibs , paragraph 2.

normally depends on many shared libraries. Those shared libraries might also be interdependent with other libraries. The *inter-libraries dependencies*[241] might also be hidden to the final developer of a program, so he could just not be aware of their license compatibilities[242] .

Also consider that most shared libraries are downloaded via default packaging systems such as *DPKG*[243] or *RPM*[244]. In such case, the developer and the packet distributor might not have the premeditation of committing copyleft infringement[245].

(2) When a non-free library depends on a GPL Program.

This is the scenario of plug-ins. There are many programs in which the plug-ins provide the program functionalities. The plug-ins are generally dependent on the program, and loaded when their functionality is required. But in many cases the plug-ins might also contain different types of code[246]. These different types of code are dependent as a Whole on the GPL program in order to function, but some of the files won't directly link to the Program. This is a very common case of famous *Content Management Systems*[247] such as *Wordpress*[248], or *Drupal*[249]. Under the dependency criteria, all the files should follow the GPL license, but in reality that is not a suitable solution. This paradigm could be solved through the application of a split GPL license[250].

241 *"An inter-library dependency is one in which a library depends on other libraries. For example, if the libtool library libhello uses the cos function, then it has an inter-library dependency on libm, the math library that implements cos"*. See, http://www.gnu.org/software/libtool/manual/html_node/Inter_002dlibrary-dependencies.html.

242 A detailed description on libraries interdependencies is available at: http://www.gnu.org/software/libtool/manual/html_node/Inter_002dlibrary-dependencies.html.

243 *"Packet manager for Debian"*. See, http://linux.die.net/man/1/dpkg.

244 *"RPM package manager"*. See, http://linux.die.net/man/8/rpm.

245 In Romano-Germanic law systems, the Intention is a general requirement for establishing liability.

246 E.g. A wordpress plug-in. A Wordpress plug-in usually contains PHP code, CSS code, HTML code. See, https://codex.wordpress.org/Writing_a_Plugin.

247 *"A content management system (CMS) is a system used to manage the content of a Web site"*. See, http://searchsoa.techtarget.com/definition/content-management-system.

248 *"WordPress is web software you can use to create a beautiful website or blog. We like to say that WordPress is both free and priceless at the same time"*. See, http://wordpress.org/.

249 *"Drupal is an open source content management platform powering millions of websites and applications"*. See, http://drupal.org/.

250 This information is expanded in the *Wordpress v Thesis* case, described in chapter 4.2 of this work.

3.3. INTERACTION

The *interaction* criterion consists of how to the program's executable connects to the shared libraries.

GPL v2: *"If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and **separate works** in themselves ..."*[251]. The program libraries and the program might be seen as separate works by themselves. The key would be to determine if they just interact through allowed function calls.

GPL v3: ***"****A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are **not combined** with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate"..."*[252] .
GPL v3 adds the combination. If interaction is considered a combination, then this provision clearly forbids interaction in the terms of linking.

GPL FAQ: *"If the program dynamically links plug-ins, and they make **function calls** to each other and share data structures, we believe they form a single program, which must be treated as an extension of both the main program and the plug-ins..."*[253].

*"**Combining two modules means connecting them together** so that they form a single larger program. If either part is covered by the GPL, the whole combination must also be released under the GPL—if you can't, or won't, do that, you may not combine them..."*[254]. The program links to the dynamic libraries by making function calls. Thus, the *copyleft* is triggered under these interpretations.

But there are exceptions: *"If the program uses fork and exec to invoke plug-ins, then the*

251 See, GPL v2 section 2, paragraph 2.
252 See, GPL v3 section 5. paragraph 2.
253 See, http://www.gnu.org/licenses/gpl-faq.html#NFUseGPLPlugins. Paragraph 2.
254 See, http://www.gnu.org/licenses/gpl-faq.html#NFUseGPLPlugins.

plug-ins are separate programs, so the license for the main program makes no requirements for them..."[255]. The fork() and exec() are common system calls in Unix environments. These system calls are used with the purpose of spawning and executing new programs[256].

There is also a borderline interpretation of interaction: *"If the program dynamically links plug-ins, but the communication between them is limited to invoking the 'main' function of the plug-in with some options and waiting for it to return, that is a borderline case"*[257]. Plug-ins might be programmed in different ways, and for different purposes. The main function might be the activation function, or the loading function. But there is a lot of uncertainty around this borderline case.

Also consider that network software interacts through sockets. Function calls are sent and received by the peers or hosts. If the GPL FAQ interpretation only applies to dynamic libraries, we should be aware that dynamic libraries could also be linked in the network. The GPL FAQ interpretation is different in network cases:

> *"By contrast, pipes, sockets and command-line arguments are communication mechanisms normally used between two separate programs. So when they are used for communication, the modules normally are* ***separate programs****. But if the* ***semantics of the communication are intimate enough,*** *exchanging complex internal data structures, that too could be a basis to consider the two parts as combined into a larger program"*[258].

ANALYSIS

Some paradigms emerge in relation to the interaction criterion: the paradigm of *libraries as computer programs*, and the paradigm of *function calls and interoperability.*

(1) Libraries as Computer Programs. Dynamic libraries can also be considered as

255 See, http://www.gnu.org/licenses/gpl-faq.html#GPLPluginsInNF. Paragraph 1.
256 An easy tutorial about system calls with the Linux kernel is available at: http://www.tuxradar.com/content/how-linux-kernel-works.
257 See, http://www.gnu.org/licenses/gpl-faq.html#NFUseGPLPlugins. Paragraph 3.
258 See, http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#TOCMereAggregation. Paragraph 4.

computer programs, or a collection of programs. They don't fuse their contents with another computer program into an executable. Some definitions consider them as programs: *"A dynamic link library (DLL) is a collection of small programs, any of which can be called when needed by a larger program that is running in the computer"*[259].

As we can see, the question here is whether dynamic libraries are considered as computer programs, or not. Technically, they are programs or collections of programs designed with the purpose of extending the functionality of other programs. Legally, they can also be understood as computer programs. The European Directive 2009/24/EC establishes: *"A computer program shall be protected if it is original in the sense that it is the* ***author's own intellectual*** *creation. No other criteria shall be applied to determine its eligibility for protection"*[260].

Following this provision, there is not a legal difference between a dynamic library and a computer program. Thus, they interact as two different computer programs. Also, there is not such difference in the GPL license. The GPL v3 defines the program as: *"The Program refers to any copyrightable work licensed under this License..."*[261]. The difference between programs and libraries was made only through the GPL FAQ interpretation. The LGPL license made such distinction, and wisely included *combined works* in its provisions. Adding *combined works* into the derived works provisions would be a good solution for the interaction paradigms in the GPL license[262].

(2) Function calls and Interoperability. The GPL FAQ interpretation accepts normal communication via sockets and pipes for the purpose of program interactivity[263]. But it also allows the possibility of normal communication for dynamic linking:

"Where's the line between two separate programs, and one program with two parts?

259 See, http://searchwinit.techtarget.com/definition/dynamic-link-library.
260 Art 1.3 Directive 2009/24/EC on the legal protection of computer programs.
261 See, GPL v3, section 0. Definitions. Available at http://www.gnu.org/licenses/gpl.html.
262 See, LGPL license v3, section 0. Definitions. Available at http://www.gnu.org/licenses/lgpl.html
263 See, http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#TOCMereAggregation. Paragraph 5.

> *This is a legal question, which ultimately judges will decide. We believe that a proper criterion depends both on the mechanism of communication (exec, pipes, rpc, **function calls** within a shared address space, etc.) and the semantics of the communication (what kinds of information are interchanged)"*[264].

The plug-ins are usually loaded by an *Application Programming Interface*[265] built within the Program. These APIs provide all the needed elements in order to connect a plug-in. The plug-in will be ready to link to the program at runtime or loading time, by the API[266]. If the API does not invoke the plug-in by default, the user has to load the plug-in.

A dynamic library, or a *plug-in* which links dynamically, usually brings some specific functions to the program. Therefore, the main function of a dynamic library is its purpose, and the *function calls* are just *connectors* which behave similar to *sockets* or pipes, because they act as *interfaces*, and allow communication between executables and dynamic libraries located elsewhere at run time or loading time.

Program libraries interaction is a common feature, and legally, it is not clear why program libraries must be treated in a different manner than computer programs. The European Directive 2009/24/EC stands: *"...The parts of the program which provide for such interconnection and interaction between elements of software and hardware are generally known as **interfaces**. This functional interconnection and interaction is generally known as interoperability"*[267].

Nevertheless, developer communities seem to accept that plug-ins of GPLed programs must be licensed under the GPL license. For example, this is the license statement that *Wordpress* recommends to plug-in developers: *"It is customary to follow the standard header with information about licensing for the Plugin. Most Plugins use the GPL2 license used by WordPress or a license compatible with the GPL2"*[268].

---

264 See, http://www.gnu.org/licenses/gpl-faq.html#MereAggregation. Paragraph 2.

265 *"An API is a set of commands, functions, and protocols which programmers can use when building software for a specific operating systems"*. See, http://www.techterms.com/definition/api.

266 This API is set by the Program for the plug-ins development, in a similar way than an Operative System offers APIs for developing Computer programs.

267 See, Recital 10 of the directive 2009/24/EC.

268 See, https://codex.wordpress.org/Writing_a_Plugin.

3.4. DISTRIBUTION MEDIUM

The *distribution medium* criterion determines if the dynamic libraries and the computer program are forced to be licensed under the GPL license, in the case that they are distributed in the same distribution medium. This criterion is not about the general distribution obligations of the GPL license such as distribution of the source code, or license distribution[269].

GPL v2: *"Mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a* ***volume of a storage or distribution medium*** *does not bring the other work under the scope of this License"*[270].

Distribution mediums are physical devices such as *DVDs*[271] or *CDs*[272], or in case of direct downloading, the equivalent is compressed packages such as *ZIP*[273] or *TAR*[274].

GPL v3: ***"****A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or* ***distribution medium****, is called an "aggregate"...*[275]*"*

The GPL v3 follows the GPL v2 distribution medium perspective. A distribution medium is not an important criteria for dynamic libraries if they are considered *aggregate works*.

GPL FAQ: *"Mere aggregation of two programs means putting them side by side on the*

269 The distribution general terms are not directly related to dynamic linked libraries. Distribution terms in GPL have a huge scope, and that is the reason they will not be confronted in this work.
270 See, GPL v2 section 2, paragraph 4.
271 See, http://en.wikipedia.org/wiki/DVD.
272 See, http://en.wikipedia.org/wiki/CD.
273 *"A zip file (.zip) is a "zipped" or compressed file"*. See, http://www.techterms.com/definition/zip.
274 *"The Tar program provides the ability to create tar archives, as well as various other kinds of manipulation"*. See, http://www.gnu.org/software/tar/.
275 See, GPL v3 section 5 paragraph 2.

same CD-ROM or hard disk. We use this term in the case where they are separate programs, not parts of a single program. In this case, ***if one of the programs is covered by the GPL, it has no effect on the other program****..."*

The GPL license makes it very clear. Mere aggregation is not combination.

ANALYSIS

The distribution medium criterion analyzed in the light of the GPL license does not present any controversy.

3.5. LOCATION (ALLOCATION)

The location criterion consists in determining if the dynamic libraries and the program are located at the same storage device or computer memory address space.

GPL v2: *"...mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a* ***volume of a storage*** *or distribution medium does not bring the other work under the scope of this License*[276]*"*. Under this provision, the volume of a storage is not relevant. But nothing is said about the *computer memory address space*.

GPL v3: *"A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on* ***a volume of a storage*** *or distribution medium, is called an "aggregate"...*[277]*"*.
Again in the GPL v3, nothing is said about the *computer memory address space*.

GNU FAQ: *"If the program dynamically links plug-ins, and they make function calls to*

276 See, GPL v2 section 2 paragraph 4.
277 See, GPL v3 section 5 par. 2.

*each other and **share data structures**, we believe they form a single program"*[278]. "*Using **shared memory** to communicate with complex **data structures** is pretty much equivalent to dynamic linking*[279]".

The *data structures*[280] of the computer memory space can be obtained, and read bit by bit[281], as it was previously indicated[282].

ANALYSIS

The location criteria can be associated with a *permanent storage location* or to a *memory space temporary allocation*. The storage location does not present any controversy, considering that the GPL license literally rejects it. If we consider that the *process* is a modified work of the libraries linked to the program, the *computer memory space* perspective gets very important because is the location where the *process* resides.

An important paradigm emerges at this point: how can we prove that an executable and some dynamic libraries were located in the computer memory address space after the computer is turned off? A procedure such as getting a *bit stream image* of the physical memory, will help to preserve the contents of the physical memory for future analysis[283]. Nevertheless, the analysis at the data structure level requires a good technical expertise[284], in special the analysis of bit-streams of *shared memory segments*[285].

---

278 See, http://www.gnu.org/licenses/gpl-faq.html#NFUseGPLPlugins . Paragraph 2.

279 See, http://www.gnu.org/licenses/gpl-faq.html#NFUseGPLPlugins . Paragraph 4.

280 "*Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address—a bit string that can be itself stored in memory and manipulated by the program*". See, http://en.wikipedia.org/wiki/Data_structure.

281 In order to do that, a bit-stream image acquisition is needed. Then, the image can be loaded in an proper computer Forensics environment.

282 All related information was already describe in chapter 2.8.

283 A brief example on how to find linked libraries inside data structures was already exposed in chapter 2.8.

284 There is a short technical description about shared memory segments in chapter 2.7. There is a technical description on ELF formats and data structures in chapters 2.3 and 2.8. However, getting deeper into these highly technical issues is out of the scope of this work.

285 *"A shared memory is an extra piece of memory that is attached to some address spaces for their owners to use. As a result, all of these processes share the same **memory segment** and have access to it".* See, http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/what-is-shm.html.

However, even if we can get a digital proof that a *copyleft* infringement happened in the memory of a computer, such infringement is not happening any more once the computer has been turned off, or the processes have been killed[286].

As a chapter conclusion, it is important to remark the importance of these criteria in order to determine under which circumstances the copyleft might be triggered in relation to *linked libraries*. The paradigms emerged from some criteria such as modification, dependency and interaction, get more relevance in legal confrontations, due to the complexity of interpreting copyleft restrictions in a particular copyright law context.

Some of these criteria are complementary regarding specific paradigms. That is the case of the *modification* and *location* criteria in relation to *the process as modified work*[287] paradigm. The fact that the *process is the modified work*, which resides temporarily in the *computer memory address space*, is the sum of the *modification* and *location* criteria.

Another case is the close relation between the *dependency* and *interaction* criteria in relation to *function calls and interoperability*[288] paradigm. The fact that function calls can be seen as mere connectors, grant their use in terms of interoperability, despite of the dependency relations between the program libraries and executables. These paradigms could be solved if the GPL would include in the future the *combined works* provisions as in the LGPL license.

Now, How to apply these criteria? In the absence of legal precedents, community disputes can be used with the purpose of understanding better these technical and legal issues of the GPL license interpretation. That is the purpose of the next chapter.

286 Indeed, this *location* paradigm is complementary to the *modification* paradigm *the process as the derivative work.* See, chapter 3.1(1) of this work.
287 See, chapter 3.1(1) of this work.
288 See, chapter 3.3(2) of this work.

CHAPTER FOUR: LINKING LIBRARIES CONTROVERSIES

The GPL license is still the most popular FOSS license. Many controversies have emerged around the GPL license and linking program libraries inside the Software developing projects. There is a constant war concerning the GPL license and the copyleft, in such a way, that the *open source software* world seems to be divided into two sides: copyleft supporters and copyleft detractors.

Despite the philosophical difference between *free software* and *open source software*[289], in practice, the *copyleft* creates a barrier in the field of derivative works between FOSS licenses. As we have already seen, the GPL is a strong copyleft license which forces the derivative works to follow the GPL license. But in the field of linking libraries, the situation is still unclear, due to the absence of case law, and a general lack of legal knowledge among developer communities.

Nevertheless, developer controversies have been happening in the last years, and certainly, many legal disputes are beginning to emerge in the field. Controversies are important for the future development of the GPL license because they provide different kinds of interpretations, and solutions adapted to real scenarios. The purpose of this chapter is to briefly discuss some of those controversies and legal disputes, and to analyze them in the light of copyright law, and the GPL FAQ interpretations of the GPL license.

4.1. THE SOLUTION: PERMISSION OF THE COPYRIGHT HOLDER

The copyright holder has the right to decide what to do with his work. But when shared libraries are used to provide some sort of functionality or dependency, it is necessary to obtain the permission for using those libraries by the correspondent copyright holder.

289 Such difference is explained by Richard Stallman. See, http://www.gnu.org/philosophy/open-source-misses-the-point.html.

This principle is not only applied to non-free libraries, because free libraries can also restrict their use due to the *viral* effect of copyleft.

For solving these issues, the GPL FAQ provides: *"If you want your program to link against a library not covered by the system library exception, you need to provide permission to do that"*[290]. This permission is not necessary when the library's license permits their use by default. This is the case of FOSS licenses with no copyleft such as the *Apache License*[291], or the *BSD License*[292].

The GPL license is a strong copyleft license which falls into the public licenses case: It is a *generic purpose license* that is not written by the copyright holder. Nevertheless the copyright holder can grant the permission to use his libraries, through writing an exception. Thus, the permission of the copyright holder is a necessary solution, when the copyright holder wants to allow others to use his GPL licensed libraries without the copyleft restrictions.

But, *why would someone license their libraries under GPL if he prefers that libraries can be used for any purpose?*, the weaker copyleft LGPL might be more suitable in such case, or non *copyleft* licenses. The answer is that conflicted issues can appear after distribution, and a license has not retroactive effects. It is possible to change license for future version releases, but not for past versions when they have been already distributed within the software[293].

As changing the GPL license for particular purposes is not an option, particular exceptions and permissions to the license must be distributed within the license. Many developer controversies have been avoided by using copyright holder exceptions. Some

290 See, http://www.gnu.org/licenses/gpl-faq.html#GPLIncompatibleLibs.
291 See, http://www.apache.org/licenses/LICENSE-2.0.
292 See, http://www.linfo.org/bsdlicense.html.
293 An interesting case is when the GNU changed their *GNU Free Document license version 1.3*, with the purpose of getting compatibility with the *Creative Commons Share alike license* adopted by Wikipedia. See, http://www.gnu.org/licenses/fdl-1.3.html, section 11.

of the most relevant in relation to the GPL license are:

(1) General Permission of the Copyright Holder: Android v Linux.

An important controversy in the Internet communities has been the Google's decision about licensing the Android OS under the *Apache 2.0 License*[294]. Android is a Linux-based Operating system developed by *Android Inc*[295]. Android Inc was supported and then bought by *Google Inc*. It was released in 2007 in cooperation with the *Open Handset Alliance*[296]. Android has become a huge success, and its portability makes it suitable for smartphones, tables, net-books, and even smart TVs. The Android system is used today in more than 150 million devices.

Linux is a kernel developed by *Linus Torvalds*[297] in 1991. The Linux kernel was released under the GPL v2 license. In the beginning, the Linux kernel was adapted for working with the GNU Operating system, forming the well-known GNU/Linux. Since then, Linux has gained an enormous popularity in many areas such as web servers, host servers, supercomputers, and in the last ten years, it has become very popular in personal computers.

Controversy: The dispute emerged because the Android OS was licensed under the Apache 2.0 license in 2008. For many free software supporters, Android had to be licensed under the GPL v2 license because it uses the Linux kernel. Another topic of discussion was if the Android OS should still be considered free software, especially because the freedom definition of Free Software Foundation was challenged by the Android developer team:

> *"Android is about **freedom** and choice. The purpose of Android is to promote openness in the mobile world, but we don't believe it's possible to predict or dictate all the uses to which people will want to put our software. So, while we encourage everyone to make devices that are open and modifiable we don't believe it is our place **to force them** to do so..."*[298].

294 See, http://www.apache.org/licenses/LICENSE-2.0.htm.
295 See, http://elinux.org/Android_History.
296 See, http://openhandsetalliance.com.
297 See, http://en.wikipedia.org/wiki/Linus_Torvalds.
298 Ref: http://source.android.com/source/licenses.html.

Despite this different definition of freedom, the Linux kernel comes with the GPL v2 license, and a general permission by Linus Torvalds.

The permission of Linus Torvalds. The copyright holder of The Linux kernel gives his permission to use his kernel in a note distributed within the GPL v2 license:

> *"NOTE! This copyright does **not** cover user programs that use kernel services by **normal system calls** – this is merely considered normal use of the kernel, and does **not** fall under the heading of derived works. Also note that the GPL below is copyrighted by the Free Software Foundation, but the instance of code that it refers to (the Linux kernel) **is copyrighted by me** and others who actually wrote it"*[299].

This permission applies to *normal system calls*. Then, the controversy turned around if the Android OS and the Linux kernel just interact by normal system calls:

Normal Interaction. The kernel uses system calls with the purpose of providing and abstraction of the hardware. Examples of normal system calls are *fork(), exec(), wait(), open(), read(),close(), socket(),* and so forth. These system calls operate in the *kernel space*, and interact with the computer programs at the *user space*[300].

Google replaced all the GNU core libraries by its own Google libraries. The well-known *GNU C library* was replaced by the *Android Bionic C library*[301], released under the BSD license. Thus, the only GPL code in Android is the Linux kernel.
The GNU position is not different: "*Google has complied with the requirements of the GNU General Public License for Linux*[302]*, but the Apache license on the rest of Android does not require source release"*. This means that the Linux kernel remains licensed under the GPL v2 License, and the Android OS was free to be licensed under other

299 See, http://kernel.org/pub/linux/kernel/COPYING.
300 A very easy introduction to the Linux kernel is available at: http://tuxradar.com/content/how-linux-kernel-works. For a complete guide about the Linux kernel, see: Hartman Greg Kroat, *Linux kernel in a nutshell*, O'reailly, United States, 2007.
301 See, http://en.wikipedia.org/wiki/Bionic_%28software%29.
302 For a complete Richard Stallman review about Android, See, https://www.gnu.org/philosophy/android-and-users-freedom.html.

licenses.

ANALYSIS

Following these arguments, it is clear enough that an operating system such as Android does not need to follow the GPL v2 license of the Linux Kernel. But, *what would happen if Linus Torvalds had not included his permission within the GPL v2 license? Can an operating system be considered a derivative work of the kernel?* The relevant criteria for determining if Android commits GPL infringement would be dependency, and interaction:

Dependency: Android depends on the Linux kernel functionalities, and the Linux kernel depends on the Android OS. But Android needs the Linux kernel considering that it has not been used with other kernels yet. However, the Linux kernel does not need Android, as it is used other operating systems such as the GNU/Linux.

Interaction: The Linux kernel is a *modular kernel*. The modules provide the kernel functionality, and they are dynamically loaded into the kernel. As we have seen, under the GPL v2 the interaction is a criterion for establishing a derivative work, but this case falls into the GPL FAQ interpretation: *"If the program uses fork and exec to invoke plug-ins, then the plug-ins are separate programs, so the license for the main program makes no requirements for them..."*[303].

In this controversy, the Linus Torvalds permission allows the Android team, or any other developer, to use the Linux kernel. Without such permission, the dependency and interaction criteria become controversial, and the only solution would be to test is the Android OS works with other kernels, and check all android libraries and source code to confirm that is not binded to the Linux kernel in more than *normal function calls.*

303 See, http://www.gnu.org/licenses/gpl-faq.html#NFUseGPLPlugins.

(2) Particular Permission for FOSS Software: MySQL v PHP

PHP is a *"widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML"*[304]. PHP is owned by the *Zend*[305] Company, and is licensed under the *PHP license*[306], a non copyleft license.

MySQL is *"the world's most popular open source database"*[307]. MySQL was originally developed by *MySQL AB*[308], then was bought by *Sun Microsystems*[309] in 2008[310], and finally acquired by *Oracle*[311] in 2010[312]. MySQL is licensed under the GPL v2 license. PHP and MySQL are widely used and distributed together in distributions such as the *LAMP*[313] server, and Apache based Web server with PHP and MySQL.

Controversy: The dispute emerged when *MySQL AB* decided to switch their libraries from the *LGPL* license to the *GPL* license in 2004. The PHP developers decided to disable the extension for MySQL libraries in PHP5. The huge PHP and MySQL *Web* developer community was very disappointed with this license conflicting issue. If PHP was not able to operate with MySQL, the result would be very negative for the open source community.

This change also affected other enterprises such as *Red Hat Enterprise Linux*[314], one of the leading Linux based companies. Red Hat as other companies were forced to use the previous versions of MySQL, or change MySQL for other databases within their distributions. Red Hat spokeswoman mentioned: *"Our core competency is not to service and support a database"*[315].

304 Copied from, http://php.net/.
305 See, http://www.zend.com/en/.
306 See, http://www.php.net/license/3_01.txt.
307 Copied from, http://www.mysql.com/.
308 See, http://en.wikipedia.org/wiki/MySQL_AB.
309 See, http://en.wikipedia.org/wiki/Sun_Microsystems.
310 See, http://www.mysql.com/news-and-events/sun-to-acquire-mysql.html.
311 See, http://www.oracle.com/index.html.
312 For a complete story about this sale, see: http://smoothspan.wordpress.com/2009/04/20/oracle-buys-mysql-java-and-some-other-stuff-now-what/.
313 See, http://www.lamphowto.com/.
314 See, http://www.redhat.com/.
315 A complete report can be found at: http://www.redhat.com/archives/taroon-list/2004-March/msg00234.html.

The FOSS license exception: To solve this license conflict, a solution emerged from MySQL AB. They released a GPL license exception, permitting the use of MySQL with the non GPL'ed software. The exception was called a *FLOSS*[316] *license exception*[317], and allowed the use of the MySQL client libraries for FLOSS software development, even if the FLOSS license is not compatible with the GPL v2 license. This license was then adopted by *Oracle*, with the name of the *FOSS license exception*.

There is a very interesting innovation to in this exception: *It is only for FOSS software*. The exception defines derivative works and FOSS applications: *"'Derivative Work" means a derivative work, as defined under applicable copyright law, formed entirely from the Program and one or more FOSS Applications". "FOSS Application" means a free and open source software application distributed subject to a license listed in the section below titled "FOSS License List"*[318].

At the end of the FOSS license exception, there is a list of all FOSS licenses that are included within the exception. The most relevant FOSS licenses are included, such as the PHP license[319], the Apache Software License[320], the BSD license[321], the MIT license[322], the EPL license[323], amongst others.

ANALYSIS

The FOSS license exception shows a different way to deal with the *copyleft*. *Oracle*, the copyright holder, included a permission for well-known FOSS licenses. Clearly, the intention is to deny the interaction of MySQL with proprietary software, in order to keep

316 "*Free and open source software*". See, http://en.wikipedia.org/wiki/Free_and_open_source_software.
317 See, http://www.mysql.com/about/legal/licensing/foss-exception/.
318 The complete Oracle's FOSS license is in section 3 of the FOSS library exception, available at: http://www.mysql.com/about/legal/licensing/foss-exception/.
319 See, http://php.net/license/index.php.
320 See, http://www.apache.org/licenses/.
321 See, http://www.linfo.org/bsdlicense.html.
322 See, http://www.linfo.org/mitlicense.html.
323 See, http://php.net/license/index.php.

it inside the *free* and *open source* world. Within the list, there are FOSS licenses with strong copyleft licenses, and non-copyleft licenses.

What would be the scenario if the MySQL exception has not been released? The panorama would get more difficult as the GPL derivative work restrictions will apply, in particular under the *modification*, *dependency*, and *interaction* criteria. PHP interacts permanently with MySQL by making function calls after connection such as:

```
<?php
mysqli_connect(host,username,password,dbname);
?>
```

and the processes are automatically generated:

```
~# sudo lsof | grep apache2

apache2   2035   www-data   mem   REG   8,4   8315616   17039512
/usr/lib/apache2/modules/libphp5.so

apache2   2035   www-data   mem   REG   8,4   121552   17043590
/usr/lib/php5/20090626+lfs/mysqli.so

~# sudo lsof | grep mysql

mysqld   1115   mysql   10u   IPv4   10082   0t0   TCP localhost:mysql (LISTEN)

….....
```

Modification: A web server and a database normally generate different processes. In the graphic we can see that the processes are different.

Dependency: PHP is *dependent* on a Database in order to function properly, but not necessarily MySQL. PHP could still connect to other databases such as *PostgreSQL*[324]. Following the dependency criterion, PHP does not exclusively need MySQL.

Interaction: The function *mysql_connect()* in PHP permits to send the user data to the MySQL database. But connecting two processes might be equal to normal interaction

324 See, http://www.postgresql.org/.

between two different computer programs. We are in the *border line case*.

In conclusion, even following the GPL FAQ interpretations of the GPL license, it is unlikely to have a copyright infringement in this particular case.
The FOSS license exception was necessary in order to avoid dealing with speculations about GPL infringement. All these restrictions would have been affected the popularity of MySQL client libraries, but the logic prevailed. The FOSS license exception constitutes another important solution based on the copyright holder's permission, which adds new important features. But it is limited only for FOSS software.

4.2. WITHOUT THE PERMISSION OF THE COPYRIGHT HOLDER

As we have seen in the previous cases, the permission of the copyright holder is by far the best solution to the GPL license dynamic libraries controversies. But when the copyright holder has not given his permission, the scenario radically changes.

Despite the lack of relevant legal precedents, many disputes have happened inside developer communities concerning the dynamic linked libraries paradigms. For the GPL supporters, *"copyleft is a general method for making a program (or other work) free, and requiring all modified and extended versions of the program to be free as well"*[325]. On the other hand, the meaning of *freedom* has been highly questioned by other open source communities, and other philosophies have emerged such as the *copyfree*[326].

The *copyleft* is not always clear, and the solutions for GPL controversies just appear when controversies emerge. In the field of linked libraries, some developer communities disputes have provided practical solutions that will have to be considered by the *Judges* in the near future. Some of them are described next.

325 See, http://www.gnu.org/copyleft/copyleft.html.
326 See, http://www.wikivs.com/wiki/Copyfree_vs_Copyleft.

(1) The Split License Solution: Wordpress v Thesis

Wordpress can be defined as: *"a free Content Management System based in PHP and MySQL"*[327]. It appeared in 2003, and since then has become the most popular blogging tool in the world. But in the last years it has also become a very popular CMS[328] for developing *Web Sites*. *Wordpress* operates in a plug-in architecture, and plug-ins provide all extra features such as *themes*[329], *language translators*, *spam blockers*[330], and so forth. *Wordpress* is licensed under the GPL v2 (or later) license.

Thesis[331] is a premium *Wordpress template theme* created by *Chris Pearson*, co-founder of a design company called *DYITthemes*[332]. The *Thesis* theme became very popular after its appearance in 2008, and became a monetary success. But his creator did not release the plug-in under the GPL v2 license. A *Wordpress theme* plug-in is often distributed as a collection of PHP[333] code, CSS[334] code, HTML[335] code, and media content.

Controversy: The controversy turned up in 2010, after an interview at the program *Mixergy*[336]. The thesis creator openly referred to the Thesis monetary success, and mentioned:*"...should have been at least partially free and open source software"*[337]. *Matt Mullenweg*[338] co-founder of *Wordpress*, immediately reacted in twitter against *Thesis*. He considered that *Wordpress* users should not use *Thesis*, because it was infringing GPL v2 license: *"We write software that empowers and protects the freedoms of users, it's our Bill of Rights. People should respect that"*[339].

---

327 See, http://en.wikipedia.org/wiki/WordPress.
328 Content Management System.
329 A *Wordpress* theme provides a design template for the WebSite. *Thesis* is one of them. See, http://diythemes.com/.
330 They block all undesired comments and emails. A very popular one is *Akismet*. See, http://akismet.com/.
331 See, http://diythemes.com/.
332 See, http://diythemes.com/.
333 *"Hypertextor Processor"*. See, http://php.net/.
334 *"Cascading Style Sheets"*. See, http://www.w3.org/Style/CSS/.
335 *"Hyper Text Markup Language"*. See, http://www.w3schools.com/html/.
336 See, See, http://mixergy.com/goto/welcome/.
337 See, http://mashable.com/2010/07/22/thesis-relents/.
338 See, http://en.wikipedia.org/wiki/Matt_Mullenweg.
339 See, http://mashable.com/2010/07/22/thesis-relents/.

Both parts were confronted in live connected by *skype*, and transmitted by *mixergy*. The interview was a battle of arguments in both directions. Matt Mullenweg mentioned: *"Thesis has stated publicly that they believe in a different interpretation of the GPL. The GPL doesn't apply to them. That Wordpress license doesn't apply and they don't need to follow it. That's obviously harmful to the Wordpress community and I would love them to join and be GPL"*[340].

The position of *Wordpress* was to denounce the copyright infringement of *Thesis*, but proposed a solution by inviting *Thesis* to join the GPL community. For him, *Thesis* was violating the GPL Derivative work's obligations. Mullenweg proposed that at least the PHP code must be licensed under the GPL v2 license.

Pearson from *Thesis*, replied: *"Thesis has over 27,000 users, many of whom were not introduced to wordPress except through Thesis … a superior functionality to that which is offered by the platform without Thesis"*[341]. For Pearson, Thesis brings a lot of good things to *Wordpress*, it was the preferred *Wordpress* theme, and brings people into *Wordpress*. Certainly, this was not a legal argument.

Mullenweg replied: *"It's just that anyone violating the license is disrespectful to thousands of people that built wordpress and all of the other businesses that have respect for wordpress license*". The arguments kept going in both directions. After some minutes Pearson came out with a good argument: *"an attorney in Florida has published an article called "Why the GPL Does Not Apply to Premium wordpress Themes"*[342]. He cited a couple of court cases as precedents, one involving *Nintendo* and the other involving the *Sega genesis console*. And, Mullenweg replied: *"...the Nintendo case, or whatever, which I think was from the 1980s or 1990s, has nothing to do with GPL..."*[343].

Finally, Brian Pearson decided to adopt Mullenweg suggestion, and adopted a *Split GPL*

340 See, http://mixergy.com/chris-pearson-matt-mullenweg/.
341 See, http://mixergy.com/chris-pearson-matt-mullenweg/.
342 See, http://perpetualbeta.com/release/2009/11/why-the-gpl-does-not-apply-to-premium-wordpress-themes/.
343 See, http://mixergy.com/chris-pearson-matt-mullenweg/.

license for Thesis. The PHP code was released under the GPL v2 license, but the graphics, the CSS and the HTML code remained under a proprietary license.

ANALYSIS

The *Wordpress v Thesis* dispute didn't become a court case. However, the split license solution provides an interesting panorama to discuss. First, *Wordpress* did not want to make a GPL exception for Thesis. *Wordpress* provides a precondition for plug-in developers: *"It is customary to follow the standard header with information about licensing for the Plugin. Most Plugins use the GPL2 license used by wordPress or a license compatible with the GPL2"*[344].

Wordpress intention seems to be that plug-ins must be licensed under the GPL2 license as a *Whole*. The final solution was to split the plug-in, and use the GPL2 license only to the PHP linked to the Application Programming Interface in the *Wordpress* platform. Independent PHP code, and another types of code such as the CSS and HTML code would not follow the GPL2 license.

What would happened if Thesis refused to adopt the GPL license? The relevant criteria would be modification, dependency and interaction.

Modification: A *Wordpress* plug-in is programmed by using the *Wordpress* plug-in API. Therefore, there is a modified work because *Wordpress* PHP classes and methods are used, copied or modified in order to create the plug-in. The GPL v2 establishes: "*You must cause any work that you distribute or publish, that* ***in whole or in part*** *contains or is derived from the Program or any part thereof, to be licensed as a whole..."*[345].

Dependency: *Wordpress* plug-ins are dependent on *Wordpress* in order to function. Plug-ins generally are dependent on the program, and this case is not an exception. Two

344 See, https://codex.wordpress.org/Writing_a_Plugin.
345 See, GPL v2, section 2, paragraph 1.b, available at, http://www.gnu.org/licenses/gpl-2.0.html.

non GPL cases of American Courts were mentioned by Pearson during the controversy, both about dependency:

- *Midway Mfg. Co. v. Artic International*[346]. This was a dispute in the video games arena, and comes from 1982. The plaintiff Midway Mfg. sued Artic International for copyright infringement in two of his proprietary games: Pacman and Galaxian. The defendant Artic International developed a set of Rom chips which improved the speed and performance of the games. The Court ruled in favor of the plaintiff considering that the copy of the images projected by the device were enough criteria to consider it as a derivative work, in the meaning of artistic works.

- *Lewis Galoob Toys, Inc. v. Nintendo of America, Inc*[347]. The defendant Lewis Galoob Toys created a special hardware add-on with the purpose of enhancing the Nintendo systems for video games. The court considered that the hardware add-on did not incorporate the protected work. This last case could be found convenient for the defense of thesis, but it was not a GPL case, and was not a hardware case.

Interaction: Thesis contained PHP, CSS and HTML code. The CSS and HTML code interact with the PHP code. But just some PHP code from thesis interacts with some PHP code in *Wordpress*. Following the Interaction criteria, the GNU interprets: *"If the program dynamically links plug-ins, but the communication between them is limited to invoking the 'main' function of the plug-in with some options and waiting for it to return that is a borderline case"*[348]. Under the Interaction criteria, this case might be a borderline case, and a Judge could also interpret it as fair use, or not subject of copyright protection.

In conclusion, in this particular case, the copyleft would be triggered by applying the modification and dependency criteria.

Apparently *Wordpress* won the dispute. But the split license solution was not harmful to Thesis, considering that the graphics, the CSS and the HTML code are a

346 See, 547 F. Supp. 999 (N.D. Ill. 1982)
347 See, 964 F.2d 965 (9th Cir. 1992)
348 See, http://www.gnu.org/licenses/gpl-faq.html#NFUseGPLPlugins. Paragraph 3.

complementary scheme, but they are the real economic value of Thesis. So in the end, Pearson did not lose, Thesis still gets economical revenue, and is still one the preferred Themes by the *Wordpress* community. The split license solution is very accurate, because just concern to the code that is actually, linked.

(2) Complete License Change: Hyper-V v Linux Drivers

This controversy emerged in 2009 when Microsoft decided to release 20000 lines of code under the GPL license. This code consisted of many Linux drivers released as Linux *Integration Component Drivers*[349]. A particularly controversial one was a network driver in a virtual server and cloud platform called Hyper-V[350]. Hyper-V driver was created using GPL libraries, and linked to proprietary code.

Controversy: Steven Hemminger leader of an open source network vendor called *Vyatta*[351], congratulated Microsoft for releasing the Hyper-V network driver for the Linux kernel. However, he discovered that the driver[352] was infringing GPL v2 licensed libraries. He told this infringement to A. Greg Kroah Hartman, executive of Novell, and he informed Microsoft about the GPL alleged infringement.

Mr. Hartman then informed *Sam Ram* a Microsoft executive about the alleged infringement. After a few days Microsoft changed the license to the GPL v2, for business reasons, based on Ramji's[353] declaration:

> *"Microsoft's decision was not based on any perceived obligations tied to the GPLv2 license. For business reasons and for customers, we determined it was beneficial to release the drivers to the kernel community under the GPLv2 license through a process that involved working closely with Greg Kroah-Hartman, who helped us understand the community norms and licensing options surrounding the drivers"*[354].

349 See, http://www.microsoft.com/en-us/download/details.aspx?id=28188.
350 See, http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx.
351 ***"Vyatta*** *manufactures software-based virtual router, virtual firewall and VPN products for Internet Protocol networks (IPv4 and IPv6)"*.See, http://en.wikipedia.org/wiki/Vyatta and http://www.vyatta.org/node/5683.
352 *"In computing, a* ***device driver*** *is a computer program that operates or controls a particular type of device that is attached to a computer".* See, http://en.wikipedia.org/wiki/Device_driver.
353 See, http://blogs.technet.com/b/port25/archive/2009/07/23/releasing-the-linux-integration-component-drivers.aspx.
354 See, http://blogs.technet.com/b/port25/archive/2009/07/23/releasing-the-linux-integration-component-drivers.aspx.

The problem was solved in a private manner, and the alleged temporal GPL infringement was rectified. The drivers were licensed under the GPL v2 license, and then included in the Linux kernel releases.

ANALYSIS

The dispute was gently solved, despite some delays, the lack of maintenance, and many doubts about Microsoft reasons[355], it seems that both parts agreed a common benefit. But this small controversy generated an immediate reaction in the FOSS community, especially considering that the principal actor was *Microsoft*. Microsoft admitted to the community that used a considerable amount of Linux code, so they accept to license them under the GPL v2[356].

What would happen if Microsoft decided not to use the GPL v2 license for the drivers?.

The modification, and dependency criteria are enough in order to answer:

Modification: The GPL FAQ interprets *"**Q:** Linux (the kernel in the GNU/Linux operating system) is distributed under GNU GPL version 2. Does distributing a nonfree driver meant to link with Linux violate the GPL?* **A:***Yes, this is a violation, because effectively this makes a larger combined work. The fact that the user is expected to put the pieces together does not really change anything"*[357].

Following this interpretation, there is a combined work. If the kernel drivers get fused with the kernel, so they definitely create a modified work. Linux components were already fused into the drivers.

Dependency: The kernel drivers were dependent on the Linux kernel.

In this case, the copyleft would be triggered by applying the modification and dependency criteria, again.

355 It seems that the delivery took very long. See, http://www.networkworld.com/news/2011/071811-microsoft-hyperv-linux.html.
356 See, http://www.kroah.com/log/linux/microsoft-linux-hyper-v-drivers.html.
357 See, http://www.gnu.org/licenses/gpl-faq.html#NonfreeDriverKernelLinux.

There is not doubt that Microsoft took the right decision. This case is somehow easy to solve due to the use of Linux components for creating the kernel drivers, and because the drivers become part of the kernel. Considering that Microsoft recognize the GPL license provisions in this driver's case, it seems that the open source communities have taken for granted that device drivers must follow the kernel's license.

4.3. COURT CASES: NON COPYRIGHT SUBJECT AND FAIR USE

A work is no subject of copyright when does not fulfill the requirements for being considered as such. The WIPO treaty stands: "*Copyright protection extends to expressions and not to ideas, procedures, methods of operation or mathematical concepts as such*"[358]. The interpretation about what is or not a subject of copyright can differ in different laws[359].

Fair use is a general copyright exception in the US law. However, there are other relevant fair use exceptions in other jurisdictions and legal systems[360]. The GPL FAQ interprets *fair use* in the following terms:

> *"Fair use is use that is allowed without any special permission. Since you don't need the developers' permission for such use, you can do it regardless of what the developers said about it—in the license or elsewhere, whether that license be the GNU GPL or any other free software license. Note, however, that there is no world-wide principle of fair use; what kinds of use are considered "fair" varies from country to country"*[361].

Fair use legally means that the user doesn't need to require the permission of the copyright holder. Common fair uses in the US are such as *reproduction for private purposes*, *teaching and illustration*, *criticism*, *citations*, parody, amongst others. In European Union law, fair use exceptions are not harmonized, but the most common are: *private copy or other private use, parody, quotation, use of a work for scientific or teaching purposes, news reporting, library privileges, needs of the administration of*

358 See, WIPO art 2. Available at http://www.wipo.int/treaties/en/ip/wct/trtdocs_wo033.html#P136_19843.
359 See, Chapter 1.3 of this work.
360 As recommended lecture for fair use in Europe, see: Hugenholtz, Senftleben, Fair use in Europe. In search of flexibbilities, Universiteit Amsterdam, The Netherlands, 2011.
361 See, http://www.gnu.org/licenses/gpl-faq.html#GPLFairUse.

justice and public policy[362].

Fair use exceptions might apply to many areas of law, included copyright law[363].

(1) Oracle v Google

This case provides an important legal precedent for the use of *program libraries*, at least in US law. *Java* is a powerful Object Oriented Programming language, initially developed by Sun Microsystems from 1990, but acquired by Oracle in 2010[364]. Java's popularity is huge among developers, and Java is a meaningful component of most contemporary software. Java and most of its dependencies migrated to the GPL v2 license between 2006 and 2007. When Oracle merged with Sun Microsystems, Java continued with the GPL v2 license. But some Java's Application Programming Interfaces come with specifications.

Understanding the dispute. Java is a powerful programming language which uses a virtual environment called the *Java virtual machine* with the purpose of running Java on different operating systems. Java is not a compiling programming language such as C, and is not an interpreter programming language such as Lisp. Better said: "*The designers of Java chose to use a combination of compilation and interpretation. Programs written in Java are compiled into machine language, but it is a machine language for a computer that doesn't really exist. This so-called "virtual" computer is known as the Java virtual machine"*[365].

Java comes with sets of class libraries for doing different kinds of jobs. Programmers can use the Application Programming Interfaces in order to get access to those class libraries[366]. A very popular Java project between Open Source developers is the *Open JDK* project which is distributed with the *ClassPath* exception and the GPL v2 license.

362 See,Dusollier Severine, *Fair Use by design in the European Copyright Directive of 2001: An Empty Promise*, University of Namur, Belgium, 2003. Page 3.

363 See, Hugenholtz, Senftleben, *Fair Use in Europe. In search of flexibilities*,Universiteit Amsterdam, The Netherlands, 2011.

364 See, http://news.cnet.com/8301-30685_3-20000019-264.html.

365 Copied from: Eck David, *Introduction to Programming using Java*, Hobart and William Smith Colleges, 1996.

366 For example, through this link there is a description of Java APIs in Java 6: http://docs.oracle.com/javase/6/docs/api/.

But the package involved on the dispute was the Java ME phone platform development, better known as the project *PhoneMe*[367], which does not come with the classPath exception. This fact makes this case relevant as a GPL legal precedent. This GPL v2 license issue was an inconvenient for Android's business model, so apparently they used the syntax of some Java APIs, and used the Java virtual machine techniques, but with their own virtual machine called the *Dalvik*[368], and wrote their own class libraries.

Controversy: The case was divided by the Judge in three parts: (1) The copyright infringement (2) patent infringement (3) Damages claim. In this work, only the copyright infringement part[369] will be analyzed.
Oracle was claiming copyright infringement over 37 Java's API packages, copying their method names and headers, and the API's structure and sequence[370]. The *Google* defensive argument was that they used Java because it is a free and open solution, and they did not copy literally the software code contained in the Java API packages, instead, Google wrote their own implementations. Considering these facts, Google argued that this was a *fair use* case.

The Judge came up with a very interesting solution. He could confirm that some of the Java's APIs code was copied, so Oracle was right. But the infringement applies only to 9 lines of code that were literally copied related to a Java function of 3179 lines called *Range Check*[371], and developed by some former Oracle programmers who today work for Google[372]. These lines were deleted of Android in 2011, when Google discovered them inside the code. So there was a very small, and perhaps accidental copy. But there

367 *"Java ME phone platform development"*. See, http://java.net/projects/phoneme.
368 See, http://en.wikipedia.org/wiki/Dalvik_%28software%29.
369 See, United States District Court for the northern district of California, *Case3:10-cv-03561-WHA* , Page 2, lines 10-17.
370 The amount of damages was considered by Oracle of nearly $6 billion. See, http://www.groklaw.net/article.php?story=20120218041255197.
371 As to certain small snippets of code, *"the jury found only one was infringing, namely, the nine lines of code called rangeCheck"* . See, United States District Court for the northern district of California, *Case3:10-cv-03561-WHA* , Page 3, lines 2-3.
372 *"In 2009, Dr. Bloch worked on Google's Android project for approximately one year. While working on the Android team, Dr. Bloch also contributed Timsort and ComparableTimsort to the Android platform. Thus, the nine-line rangeCheck function was copied into Google's Android. This was how the infringement happened to occur"*. See, United States District Court for the northern district of California, , *Case3:10-cv-03561-WHA*, Page 13, lines 18-24.

was not a *causal nexus* between the copied code, and the alleged infringement.

The exchange of arguments between Oracle's lawyer and the Judge was very interesting. The plaintiff Oracle argued: "*I still think it's possible to demonstrate a nexus by showing that speed was very important to Google in getting Android out, and by copying they accelerated that*"[373].

And the Judge replied:

> "*I have done, and still do, a significant amount of programming in other languages. I've written blocks of code like rangeCheck a hundred times before. I could do it, you could do it. The idea that someone would copy that when they could do it themselves just as fast, it was an accident. There's no way you could say that was speeding them along to the marketplace. You're one of the best lawyers in America, how could you even make that kind of argument?*"[374].

Following the interpretation criteria of the United States copyright Act, 9 lines from 3179 were not a substantial proportion. But the plaintiff replied:

"*I want to come back to rangeCheck".* And the Judge said: "*RangeCheck! All it does is make sure the numbers you're inputting are within a range, and gives them some sort of exceptional treatment... a high school student could do it*"[375]. In respect to those 9 lines of code, the code in Range Check also did not have a considerable market effect, therefore, there was not a causal nexus between the alleged six billion dollars, and those 9 lines of code.

About the header and method names, the jury found they were not subject of copyright protection under US Copyright law: "*Copyright law does not protect names, titles, or short phrases or expressions. Even if a name, title, or short phrase is novel or distinctive or lends itself to a play on words, it cannot be protected by copyright*"[376].

About the structure of the API's, the resolution was:

> "*Each command calls into action a pre-assigned function.* ***The overall name tree, of course, has creative elements but it is also a precise command structure*** *— a*

373 See, http://www.i-programmer.info/news/193-android/4224-oracle-v-google-judge-is-a-programmer.html.
374 See, http://www.i-programmer.info/news/193-android/4224-oracle-v-google-judge-is-a-programmer.html.
375 See, http://www.i-programmer.info/news/193-android/4224-oracle-v-google-judge-is-a-programmer.html.
376 U.S. Copyright Office, Circular 34; see 37 C.F.R. 202.1(a).

> *utilitarian and functional set of symbols, each to carry out a pre-assigned function. This command structure is a system or method of operation under Section 102(b) of the Copyright Act and, therefore, cannot be copyrighted Duplication of the command structure is necessary for interoperability"*[377].

The Judge found that the method and header names, and the particular API's structures and sequences are not subject of copyright protection, but he did not establish a general legal precedent, as he clearly wrote in his conclusion:

> *"This order does not hold that Java API packages are free for all to use without license.* ***It does not hold that the structure, sequence and organization of all computer programs may be stolen****. Rather, it holds on the specific facts of this case, the particular elements replicated by Google were free for all to use under the Copyright Act. Therefore, Oracle's claim based on Google's copying of the 37 API packages, including their structure, sequence and organization is DISMISSED..."*[378].

ANALYSIS

The Judge considered that the method names and the APIs structures sued by Oracle were not subject of copyright in this particular case. The controversial 9 lines of the *RangeCheck* function were not enough due to their procedural content, and the very small amount of code that they represent in comparison to the 3179 lines of code of the API packages.

This case is relevant for various reasons regarding the interpretation of the GPL license: (1) The free version of the Java programming language called *Open JDK* is licensed under the GPL v2 license with the *classPath exception*[379]. But the *Phoneme* project did not include a *classPath exception*. However, the Judge did not even mention the GPL v2 license. There was a big expectation about other legal issues, in special if the use of APIs must be considered as fair use, as Google argued.

(2) Application Programming Interfaces extend the functionality of OOP languages, otherwise they are useless. As the statements of findings in this case mentioned: *"An*

377 See, United States District Court for the northern district of California, Case3:10-cv-03561-WHA , Page 4, lines 7-11.
378 See, United States District Court for the northern district of California, *Case3:10-cv-03561-WHA* , Page 41, lines 6-11. Decision was issued the 12.05.2012.
379 See, http://openjdk.java.net/legal/gplv2+ce.html.

API is like a library. Each package is like a bookshelf in the library. Each class is like a book on the shelf. Each method is like a how-to-do-it chapter in a book. Go to the right shelf, select the right book, and open it to the chapter that covers the work you need"[380]. But in the end, the Judge did not solve if the use of APIs should be considered *fair use*. He specified that this is a particular interpretation based on the facts of this case, therefore, it should not be applied as general rule.

This case has an important relevance for future GPL disputes, because the solution is proportional. There were 9 lines of code of potential copyright infringement, but those lines were just not relevant enough. This case makes very clear that interpreting the law is a competence of Judges.

The GPL FAQ interpretations are somehow important because they show what did the license writer meant with the GPL license. They can be taken as a guide for developers and users. But they do not mention principles such as proportionality, or flexibility. Those principles are somehow avoided, and delegated to the Courts.

(2) SAS v WPL

This is not a GPL related case, but it provides a legal precedent in Europe. In order to complement the *Oracle v Google* analysis, it will be briefly discussed. *Statistical Analysis System*(SAS)[381] is the leading *business analytic software* the market. It is maintained by SAS Institute Inc. World Programming System(WPS)[382] is a similar statistical program developed by World Programming Limited.

A statistical Analysis System provides automatic processes for the analysis of data sets. It performs functions such as *calculation* of *central tendency, frequency distribution,*

380 See, United States District Court for the northern district of California, Case3:10-cv-03561-WHA , Page 5, lines 16-23.
381 See, http://www.sas.com/.
382 See, http://www.teamwpc.co.uk/products/wps.

and association[383]. Statistical Software such as SAS permits processing massive amounts of data, and use it for practical purposes.

Controversy: SAS sued WPS for copyright infringement[384]. They claimed that WPS acquired copies of the learning edition of SAS with the purpose of developing a similar Software System due to the high demand in the market. WPS studied the functioning of the SAS software, but did not copy any part of the source code. SAS sued WPS in the High Court in the United Kingdom. The charge was a copyright infringement of WPS for copying SAS manuals and components.

The Court resolved in favor of WPS because the program functionality is not subject of copyright protection based on the scope of the Directive 2009/24/EC. The fact that WPS copied SAS functionality was not an infringement, just the *expression that comes out of that functionality* would be subject of protection. The expression could consist about *copying* and *transforming* the source code of SAS, for creating a new program.

The Directive rules: "*A computer program shall be protected if it is original in the sense that it is the author's own intellectual creation. No other criteria shall be applied to determine its eligibility for protection"*[385].
But there is a restriction: "*Ideas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright under this Directive"*[386].

ANALYSIS

This is the precedent that leaves this case: "*The functionality of a computer program or programming language is not subject of copyright"*[387]. WPL copied the functionality of SAS, but did not copy the source code.

383 See, http://www.wisegeek.com/what-is-a-statistical-analysis-system.htm
384 See, ECJ_C406_10.
385 See, Directive 2009/24/EC on the legal protection of computer programs.
386 See, Art 1.2 Directive 2009/24/EC on the legal protection of computer programs.
387 See, Court of Justice of the European Union, *Case C-406/10*, Press release No 53/12, 2012.

If we compare this case to the *Oracle v Google* case, they have similarities and differences. They are similar because both decided in favor of the defendant due to the non copyright condition of the alleged source code. But the difference is that Google actually copied and used the Java APIs. WPL did not have access to the SAS source code, they just studied how the program functions.

As a chapter conclusion, we must consider that if programming languages and their APIs are not subject of copyright protection, then in many cases, dynamic linked libraries disputes may become irrelevant. Unfortunately, the GPL license must be interpreted inside the boundaries of copyright law, and if Judges do not consider the GPL FAQ interpretations as appropriates, there is nothing to do about it.

4.4. AND THE FUTURE...

Free Software started as a dream in the eighties, and we are living that dream today. Many of us have received a huge benefit from this dream, especially in terms of free knowledge and development. This is why we should find other perspectives about how to interpret derivative works in the GPL license.

Many interpretation issues has been evolved in the last decade. Ten years ago Lawrence Rosen said: *"Many users of open-source software are frightened by the term "derivative works. They worry they might accidentally create derivative works and put their own proprietary software under an open-source license"*[388]. There was already an awareness about the technical difficulties of the GPL interpretation of derivative works in a copyright law context.

In the other side for authors such as Eben Moglen, he mentioned in 2001 that Free Software was just an unsual concept for contemporary society :

388 Rosen Lawrence, *Derivative works*, Linux journal, 2003. See, http://www.linuxjournal.com/article/6366.

> *"Because free software is an unorthodox concept in contemporary society, people tend to assume that such an atypical goal must be pursued using unusually ingenious, and therefore fragile, legal machinery. But the assumption is faulty. The goal of the Free Software Foundation in designing and publishing the GPL, is* ***unfortunately unusual****: we're reshaping how programs are made in order to give everyone the right to understand, repair, improve, and redistribute the best-quality software on earth"*[389].

But today, there is a general conscience that the technical and legal problems concerning the old copyright law framework and emerging legal challenges, have to be solved, as *Malcolm Bain* wrote in respect of the work of the *European Legal Network of Lawyers: "More practically, the Document is a work in progress, and we need more examples and/or diagrams that can help understand the technical issues involved (using header file data, published APIs, etc.) - something that might even be used as a model for presenting and arguing a case either between parties or before the courts"*[390].

In my opinion, even if the GPL FAQ interpretation is sometimes considered not flexible, and not always harmonized with copyright laws, it is also showing to the *copyright world* that the peculiarities of the *software* have to be treated in a more suitable manner. It is time to stop treating software as literary works, because they are not the same. In contemporary software, computer programs are never independent, they are built on top of programming languages, using *application programming interfaces*, and linked with shared libraries in order to adopt functionalities. All these features are beyond the scope of literary works.

There are paradigms to be solved, and some of them have been confronted here. But despite these paradigms, we can testify how the GPL license is expanding copyright law conceptions regarding software. **In the end, the GPL license is making a huge contribution to the evolution of copyright law, and that is infinitely valuable.**

389 See, http://www.gnu.org/philosophy/enforcing-gpl.html.
390 This work is called *"Software Interactions and the GNU General Public License"* and is available at http://www.ifosslr.org/ifosslr/article/view/44/74.

CONCLUSIONS

1. Copyright law must be updated. The GPL license shows how outdated is copyright law. The Berne Convention inspired most copyrights laws around the planet, but emerging new environments such as software, are in particular, free software, are far beyond the Berne Convention regulations.

2. Software and literary works are not the same. Contemporary software presents different paradigms in relation to linking program libraries, and the GPL FAQ interpretation interprets something that copyright law only regulates from a very generic perspective. Copyright laws should include *Combined works* into their derivative work provisions regarding software, and stop regulating Software as literary works, because they are not the same[391].

3. The copyleft has a purpose for derivative works. In contemporary software, licenses with strong copyleft such as the GPL, have created a barrier in the field of derivative works. The GPL follows the GNU project philosophy of free software in concordance with the four basic free software freedoms[392]. This philosophy appeared when all software was proprietary. The biggest strength of the GPL license is still its *free as freedom* perspective, and if many developers still choose the GPL license, is because they want to. Developers can always choose a non copyleft license, or other philosophy such as the *copyfree*[393].

4. Program libraries are a particular category. Program libraries should have a special legal treatment in the field of derivative works. They provide functionality and to fulfill that purpose, they need to link with computer programs[394]. They are computer

391 See, chapter 4.4: *"And the future..."*.
392 See, chapter 1.4 of this work: *"Brief GNU and GPL history"*.
393 See, chapter 4.2:*"Without the permission of the copyright holder"*.
394 See, chapter 2.1: *"Program libraries"*.

program components, and programs by themselves[395]. The GPL criteria for establishing libraries as derivative works turn around modification, dependency, interaction, distribution medium and location[396].

5. The GPL license is not focused in program libraries. The GPL FAQ interpretation might be considered too strict by some developers. Certainly, the GPL license is not focused on program libraries. That was the reason why the GNU project created the LGPL license[397].

6. The Copyright Holder can make his own exceptions. Under copyright laws, the solution is always in the hands of the copyright holders. Many exceptions have been made such as the *classPath* exception[398], the Oracle FOSS license exception[399], or the Linux kernel permission[400]. The text of generic purpose licenses such as GPL cannot be changed, but it is possible to add permissions and exceptions with an extra clause, and distribute this clause within the GPL license.

7. The problem is license compatibility. Developers can find very difficult the task of using dynamic libraries because of the license compatibility. A program can use many dynamic libraries, and all of them have to be compatible. The final purpose of the Free Software Foundation is to separate as much as possible the free software world from the proprietary software world[401]. In the end, copyright holders decide.

8. The GPL code should concern only GPL code. A plug-in or a dynamic library is not always a *whole*. A library might content tons of object code, and other non linked code. These codes might be developed and licensed by different authors. If the library is linked to a GPL licensed program, then just the linked object code should be concerned

395 See, chapter 3.3.(1): *"Libraries as computer programs"*.
396 See, chapter 3: *"The GPL FAQ interpretation of linked libraries"*.
397 See, chapter 1.7(1): *"Lesser GNU General Public License"*.
398 See, chapter 1.7(3): *"The GNU classPath exception"*.
399 See, chapter 4.1(2): *"Particular permission for FOSS software: MySQL v PHP"*.
400 See, chapter 4.1.(1): *"General permission of the copyright holder: Android v Linux"*.
401 See, chapter 1.4: *"Brief GNU and GPL history"*.

about the GPL, and not all the package because that situation might result unfair for other developers. The GPL split solution[402] is a good approach.

9. Dynamic libraries and Executables don't form a whole.

If Dynamic libraries are not included in the executable, the only possible derivative work is a process allocated in the computer memory space[403]. This process will be launched at runtime or loading time by the user, not by the developer. In such case, there might be a *legal bug* in the GPL license because if the user generally runs the program for private purposes, it would be a matter of interpretation to determine is a real copyleft infringement[404] is possible. This uncertainty might be solved by the Copyright holder through making clear if he allows or not dynamic linking.

10. Interaction is granted by copyright law. Linking libraries is the process of connecting computer programs with dynamic libraries by making function calls. With dynamic libraries, this interaction is similar than the interaction between separate computer programs by sockets[405]. If applicable copyright law does not make a difference between computer programs and computer libraries, normal function calls might still be considered under the interoperability software permissions.

11. Legal exceptions can invalidate GPL provisions. Each legal system follows its own criteria for determining situations such as which works are subject or not of copyright protection[406]. Fair uses and public policies might also be important in terms of proportionality and logical sense[407]. Developers should be aware that the GPL FAQ interpretation is not the final word, and the GPL license can be interpreted in different ways[408].

402 See, chapter 4.2(1): *"Wordpress v Thesis"*.
403 See, chapter 2.8: *"Computer Memory Address Space"*.
404 See, chapter 3.1(1): *"The process as modified work"*.
405 See, chapter 3.3(2): *" Function calls and interoperability"*.
406 See, chapter 1.3: *"National copyright laws"*.
407 See, chapter 4.3: *"Non copyright subject and fair use"*.
408 See, chapter 4.3: *"Court cases: Non copyright subject and fair use"*.

12. Application Programming Interfaces are libraries. APIs are fundamental in Object Oriented Programming, and they are libraries[409]. If APIs are not considered as subject of copyright protection, then most OOP dynamic linking controversies might become irrelevant[410].

13. More legal precedents are needed. Until now, most of the controversies concerning the use of dynamic linked libraries have been resolved inside the communities, through *online* confrontations, or just by default. Nevertheless, as Court cases are beginning to appear, the tendency is that they will exponentially increase in the near future. Those precedents are certainly needed for deploying a more concrete line to follow towards the GPL license in the field of derivative works.

409 See, chapter 2.6: *"Object Oriented Programming"*.
410 See, chapter 4.3(1): *"Oracle v Google"*.

BIBLIOGRAPHY

BOOKS, LEGAL TEXTS AND ACADEMIC WORKS:

- Aquilina, Casey, Maley, *Malware Forensics*, Syngress publishing Inc, United States, 2008.
- Bain Malcolm, *Software interaction and the GNU General Public License*, International Free and Open Source Software Law Review, Barcelona, 2010.
- Bovet, Cesati, *Understanding the Linux Kernel 3rd edition*, O'reilly, United States, 2006.
- Carrier Brian, *File System Analysis*, Addison Weshley Professional, United States, 2005.
- Detterman Lothar, Dangerous Liaisons – Software combinations as derivative works?, Berklee Technology Law journal, United States, 2006.
- Drossopoulou, Eisenbach, *Manifestations of Java Dynamic Linking – an approximate understanding at source language level*-, Imperial College, London, 2002.
- Dusollier Severine, *Fair Use by design in the European Copyright Directive of 2001: An Empty Promise*, University of Namur, Belgium, 2003.
- Eck David, *Introduction to Programming using Java*, Hobart and William Smith Colleges, 1996.
- Eckel Bruce, *Thinking in C++ vol1*, United States, Prentice Hall Inc, 2000.
- Eckel Bruce, *Thinking in Java fourth edition*, United States, Prentice Hall Inc, 2006.
- Enriquez Luis, *"Android" and the GPL license: Is the Linux-adaptation by Google still free software?*, University of Hanover, Germany, 2012.
- European Court of Justice, *ECJ_C406_10 , SAS v WPL*, Luxembourg, 2012.
- European Parlament and the Council of the European Union, *Directive 2001/29/EC on the harmonisation of certain aspects of copyright and related rights in the information society* , Official Journal of the European Communities, 2001.
- European Parlament and the Council of the European Union, Directive 2009/24/EC on the legal protection of computer programs , Official Journal of the European Communities, 2009.
- Free Software Foundation, *GPL FAQ*, http://www.gnu.org/licenses/gpl-faq.
- Free Software Foundation, *Shared library support for GNU*, Free Software Foundation, United States, 1996.
- Free Software Foundation, *GNU General Public License v2*, United States, 1991. http://www.gnu.org/licenses/gpl-2.0.html
- Free Software Foundation, *GNU ClassPath exception*, United States, 2006. http://www.gnu.org/software/classpath/license.html
- Free Software Foundation, *GNU General Public License v3*, United States, 2007. http://www.gnu.org/licenses/gpl.html

- Free Software Foundation, *GNU Lesser General Public License v3*, United States, 2007. http://www.gnu.org/licenses/lgpl.html
- Garren Scott, *Copyright Protection of Computer Software: History, Politics, and Technology*, Massachusetts Institute of Technology, United States, 1991.
- German Bundestag, *Urheberrechtsgesetz – UrhG*, Germany, 1965, amended in 1998.
- Hartman Greg Kroat, *Linux kernel in a nutshell*, O'reailly, United States, 2007.
- Hugenholtz, Senftleben, Fair use in Europe. In search of flexibilities, Universiteit Amsterdam, The Netherlands, 2011.
- Jaeger Till, *Kommerzielle Applikationen für Open Source Software und deutsches Urheberrecht* , Ifross, Stuttgart, 2008.
- Kleiman Dave, The official CHFI Exam 312-49 study guide, Syngress Publishing, United States, 2007.
- Laurent Philippe, *Free and Open Source Software Licensing: a reference for the reconstruction of "virtual commons"?*, University of Namur, Belgium, 2010.
- Oracle, *Oracle's FOSS license exception*, *United States*, *http://www.mysql.com/about/legal/licensing/foss-exception/* , 2009.
- Parlement Français, *Code de la proprieté intellectuelle*, France, 1992.
- Phillips Douglas, *The Software license unveiled*, Oxford University Press, New York, 2009.
- Rosen, Lawrence, *Open Source Licensing Software Freedom and Intellectual Property Law* , Prentice Hall, United states, 2004.
- Salzman, Burian, Pomerantz, *The Linux Kernel Module Programming guide*, Peter Jay Salzman, 2001.
- Sedgewick Robert, *Algorithms in C*, Addison-Wesley Publishing, United States, 1990.
- United States 94th congress, *US copyrigth act of 1976: title 17*, United States, 1976.
- United States District Court for the northern district of California, *Case3:10-cv-03561-WHA*, United States, 2012.
- Van den Branden, Coughlan, Jaeger, *The International Free and Open source Book*, Open source Press, Germany, 2011.
- Van Holst Walter, *Copyleft, -right and the case law on APIs on both sides of the Atlantic*, International Free and Open Source Software Law Review, Netherlands, 2013.
- Vostokov Dmitri, *Memory Dump Analysis Anthology volumen 1*, Open task, Ireland, 2008.

BLOGS AND TUTORIALS:

- Duarte Gustavo, *Anatomy of a program in memory*, http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory.Free Software Foundation, *libraries-dependencies*, http://www.gnu.org/software/libtool/manual/html_node/Inter_002dlibrary-dependencies.html.Honeynet Project, Open Source Licensing Madness, http://www.honeynet.org/node/879.

- Gee Sue, Oracle v Google Judge is a Programmer!, http://www.i-programmer.info/news/193-android/4224-oracle-v-google-judge-is-a-programmer.html.
- Kerrisk Michael, Dynamic linking and derivative works, http://lwn.net/SubscriberLink/548216/07d6e8ede242237c/.
- Cnet.com, *MySQL addresses open-source license problem*, 2004, http://news.cnet.com/2100-7344_3-5173014.html?tag=nefd_top.
- Linus Torvalds, *Linux kernel permission*, https://www.kernel.org/pub/linux/kernel/COPYING.
- Linux tutorials, *static, Shared dynamic and loadable linux libraries*, http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html .
- Linux Howtos, *Sockets tutorial*, http://www.linuxhowtos.org/C_C++/socket.htm.
- Mixergy, *Would Wordpress sue the maker of Thesis, a leading Wordpress theme?, with Chris Pearson and Matt Mullenweg*, 2010, http://mixergy.com/chris-pearson-matt-mullenweg/.
- Network world, *Microsoft struggles to get Hyper-V drivers in Linux kernel*, 2011, http://www.networkworld.com/news/2011/071811-microsoft-hyperv-linux.html.
- O'Dell Jolie, *Wordpress theme Thesis maker backs down, adopts GPL*, 2010, http://mashable.com/2010/07/22/thesis-relents/.
- Ramji Sam, Releasing the Linux integration component drivers, 2009, http://blogs.technet.com/b/port25/archive/2009/07/23/releasing-the-linux-integration-component-drivers.aspx.
- Rosen Lawrence, *Derivative works*, Linux journal, 2003. See, http://www.linuxjournal.com/article/6366.
- Stallman Richard, What is Copyleft?, 1996, http://www.gnu.org/copyleft/copyleft.html.
- Stallman Richard, *Why Open source misses the point of Free Software*, 2007, http://www.gnu.org/philosophy/open-source-misses-the-point.html.
- TuxRadar Linux, How the Linux kernel works, 2009, http://tuxradar.com/content/how-linux-kernel-works.
- University of Washington school of Law, The GPL and Derivative Works, http://www.law.washington.edu/lta/swp/Law/derivative.html.
- University of Washington school of Law, *Contracts or licenses: Does it matter?*, http://www.law.washington.edu/lta/swp/Law/contractvlicense.html.
- Wikivs, *Copyfree v Copyleft*, http://www.wikivs.com/wiki/Copyfree_vs_Copyleft.

LIST OF ABBREVIATIONS

Abbreviation	Meaning	Page
WIPO	World Intellectual Property Organization	1
GNU	GNU's not Unix	9
GCC	GNU Compiler Collection	9
GDB	GNU Debugger	9
FSF	Free Software Foundation	9
FOSS	Free and Open Source Software	11
DRM	Digital Rights Management	12
LGPL	Lesser General Public License	17
API	Application Programming Interface	29
JVM	Java Virtual Machine	39
HTTP	Hyper Text Transfer Protocol	41
P2P	Peer to Peer	42
RAM	Random Access Memory	43
ELF	Executable and Link Format	45
NTFS	New Technologies File System	47
FAQ	Frequently Asked Questions	48
OOP	Object Oriented Programming	51
PHP	PHP Hypertext Preprocessor	69
SQL	Structured Query Language	69
FLOSS	Free Libre and Open Source Software	70

ABOUT THE AUTHOR

Luis Enríquez. Copyright, Copyleft and Media lawyer. I did my formal education in Information Technology and Intellectual property Law(LLM - Leibniz Universität Hannover – Germany), International Economic Law(LLM - Univesidad Andina Simón Bolívar – Ecuador), Digital Forensics and Ethical Hacking(CHFI && CEH - ECCouncil), Algorithm Composition and Sonology(Royal Conservatoire – The Netherlands)… And my informal education in file sharing sites and Internet Forums for the last 15 years.

As a FOSS lawyer and a computer security researcher, my passion is to work directly with software developers and digital artists, talking in their 'hi-tech' language and solving their legal problems. In the other hand, I also work with lawyers, talking in their 'legal' language, and solving their technical problems. Don't hesitate to contact me:

email: luisenriquez@geek4nongeeks.com luisenriquez@fosslawyers.org